# Red Hat Certificate System 8.0 Deployment Guide

Deploying Red Hat Certificate System 8.0
Edition 8.0.5

Landmann

# Red Hat Certificate System 8.0 Deployment Guide

---

## Deploying Red Hat Certificate System 8.0
Edition 8.0.5

Landmann
rlandmann@redhat.com

### Legal Notice

Copyright © 2009 Red Hat, Inc..

This document is licensed by Red Hat under the Creative Commons Attribution-ShareAlike 3.0 Unported License. If you distribute this document, or a modified version of it, you must provide attribution to Red Hat, Inc. and provide a link to the original. If the document is modified, all Red Hat trademarks must be removed.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, MetaMatrix, Fedora, the Infinity Logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack Logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

### Abstract

This guide is intended for experienced system administrators planning to deploy the Certificate System. Certificate System agents should refer to the Certificate System Agent's Guide for information on how to perform agent tasks, such as handling certificate requests and revoking certificates. For information on using Certificate System to manage smart cards and security tokens, see Managing Smart Cards with the Enterprise Security Client.

# Table of Contents

# About This Guide

This guide explains how to install and configure Red Hat Certificate System subsystems.

This guide is intended for experienced system administrators planning to deploy the Certificate System. Certificate System agents should refer to the *Certificate System Agent's Guide* for information on how to perform agent tasks, such as handling certificate requests and revoking certificates. For information on using Certificate System to manage smart cards and security tokens, see *Managing Smart Cards with the Enterprise Security Client*.

Before using Certificate System, become familiar with the following concepts:

- Intranet, extranet, and Internet security and the role of digital certificates in a secure enterprise, including the following topics:
  - Encryption and decryption
  - Public keys, private keys, and symmetric keys
  - Significance of key lengths
  - Digital signatures
  - Digital certificates, including different types of digital certificates
  - The role of digital certificates in a public-key infrastructure (PKI)
  - Certificate hierarchies
- LDAP and Red Hat Directory Server
- Public-key cryptography and the Secure Sockets Layer (SSL) protocol, including the following:
  - SSL cipher suites
  - The purpose of and major steps in the SSL handshake

## 1. Examples and Formatting

### 1.1. Formatting for Examples and Commands

All of the examples for Red Hat Certificate System commands, file locations, and other usage are given for Red Hat Enterprise Linux 5 (32-bit) systems. Be certain to use the appropriate commands and files for your platform.

> **Example 1. Example Command**
>
> To start the Red Hat Certificate System:
>
> ```
> service pki-ca start
> ```

### 1.2. Tool Locations

All of the tools for Red Hat Certificate System are located in the **`/usr/bin`** directory. These tools can be run from any location without specifying the tool location.

### 1.3. Guide Formatting

Certain words are represented in different fonts, styles, and weights. Different character formatting is used to indicate the function or purpose of the phrase being highlighted.

| Formatting Style | Purpose |
|---|---|
| **`Monospace font`** | Monospace is used for commands, package names, files and directory paths, and any text displayed in a prompt. |
| `Monospace with a background` | This type of formatting is used for anything entered or returned in a command prompt. |
| *Italicized text* | Any text which is italicized is a variable, such as *instance_name* or *hostname*. Occasionally, this is also used to emphasize a new term or other phrase. |
| **`Bolded text`** | Most phrases which are in bold are application names, such as **Cygwin**, or are fields or options in a user interface, such as a **`User Name Here:`** field or **Save** button. |

Other formatting styles draw attention to important text.

**NOTE**

A note provides additional information that can help illustrate the behavior of the system or provide more detail for a specific issue.

**IMPORTANT**

Important information is necessary, but possibly unexpected, such as a configuration change that will not persist after a reboot.

**WARNING**

A warning indicates potential data loss, as may happen when tuning hardware for maximum performance.

## 2. Additional Reading

The documentation for Certificate System includes the following guides:

- *Certificate System Deployment Guide* describes basic PKI concepts and gives an overview of the planning process for setting up Certificate System.

  This manual is intended for Certificate System administrators.
- *Certificate System Installation Guide* covers the installation process for all Certificate System subsystems.

  This manual is intended for Certificate System administrators.
- *Certificate System Administrator's Guide* explains all administrative functions for the Certificate System. Administrators maintain the subsystems themselves, so this manual details backend configuration for certificate profiles, publishing, and issuing certificates and CRLs. It also covers managing subsystem settings like port numbers, users, and subsystem certificates.

  This manual is intended for Certificate System administrators.

- *Certificate System Agent's Guide* describes how agents — users responsible for processing certificate requests and managing other aspects of certificate management — can use the Certificate System subsystems web services pages to process certificate requests, key recovery, OCSP requests and CRLs, and other functions.

  This manual is intended for Certificate System agents.
- *Managing Smart Cards with the Enterprise Security Client* explains how to install, configure, and use the Enterprise Security Client, the user client application for managing smart cards, user certificates, and user keys.

  This manual is intended for Certificate System administrators, agents, privileged users (such as security officers), and regular end users.
- *Using End User Services* is a quick overview of the end-user services in Certificate System, a simple way for users to learn how to access Certificate System services.

  This manual is intended for regular end users.
- *Certificate System Command-Line Tools Guide* covers the command-line scripts supplied with Red Hat Certificate System.

  This manual is intended for Certificate System administrators.
- *Certificate System Migration Guide* covers version-specific procedures for migrating from older versions of Certificate System to Red Hat Certificate System 8.0.

  This manual is intended for Certificate System administrators.
- *Release Notes* contains important information on new features, fixed bugs, known issues and workarounds, and other important deployment information for Red Hat Certificate System 8.0.

All of the latest information about Red Hat Certificate System and both current and archived documentation is available at http://www.redhat.com/docs/manuals/cert-system/.

## 3. Giving Feedback

If there is any error in this *Deployment Guide* or there is any way to improve the documentation, please let us know. Bugs can be filed against the documentation for Red Hat Certificate System through Bugzilla, http://bugzilla.redhat.com/bugzilla. Make the bug report as specific as possible, so we can be more effective in correcting any issues:

- Select the Red Hat Certificate System product.
- Set the component to **`Doc - deployment-guide`**.
- Set the version number to 8.0.
- For errors, give the page number (for the PDF) or URL (for the HTML), and give a succinct description of the problem, such as incorrect procedure or typo.

  For enhancements, put in what information needs to be added and why.
- Give a clear title for the bug. For example, **`"Incorrect command example for setup script options"`** is better than **`"Bad example"`**.

We appreciate receiving any feedback — requests for new sections, corrections, improvements, enhancements, even new ways of delivering the documentation or new styles of docs. You are welcome to contact Red Hat Content Services directly at docs@redhat.com.

## 4. Document History

| Revision 8.0.4 | September 24, 2009 | Ella Deon Lackey |
|---|---|---|

Tech edits to the subsystems overview chapter, per Bugzilla #510596. This completes the tech reviews for this guide.

| | | |
|---|---|---|
| **Revision 8.0.3** | **September 19, 2009** | **Ella Deon Lackey** |

Further edits to the key length and strength section in chapter 1, per feedback from Bob Relyea.

| | | |
|---|---|---|
| **Revision 8.0.2** | **September 9, 2009** | **Ella Deon Lackey** |

Tech edits to chapter 1, per Bugzilla #510594.

| | | |
|---|---|---|
| **Revision 8.0.1** | **August 5, 2009** | **Ella Deon Lackey** |

Tech edits to chapters 3, 4, and 5, per Bugzilla #510598, #510599, #510597. The biggest changes were to the SELinux feature overview.

| | | |
|---|---|---|
| **Revision 8.0.0** | **July 22, 2009** | **Ella Deon Lackey** |

Initial draft for Certificate System 8.0 *Deployment Guide*.

# Chapter 1. Introduction to Public-Key Cryptography

Public-key cryptography and related standards underlie the security features of many products such as signed and encrypted email, single sign-on, and Secure Sockets Layer (SSL) communications. This chapter covers the basic concepts of public-key cryptography.

Internet traffic, which passes information through intermediate computers, can be intercepted by a third party:

- *Eavesdropping.* Information remains intact, but its privacy is compromised. For example, someone could gather credit card numbers, record a sensitive conversation, or intercept classified information.
- *Tampering.* Information in transit is changed or replaced and then sent to the recipient. For example, someone could alter an order for goods or change a person's resume.
- *Impersonation.* Information passes to a person who poses as the intended recipient. Impersonation can take two forms:
  - *Spoofing.* A person can pretend to be someone else. For example, a person can pretend to have the email address **`jdoe@example.net`** or a computer can falsely identify itself as a site called **`www.example.net`**.
  - *Misrepresentation.* A person or organization can misrepresent itself. For example, a site called **`www.example.net`** can purport to be an on-line furniture store when it really receives credit-card payments but never sends any goods.

*Public-key cryptography* provides protection against Internet-based attacks through:

- *Encryption and decryption* allow two communicating parties to disguise information they send to each other. The sender encrypts, or scrambles, information before sending it. The receiver decrypts, or unscrambles, the information after receiving it. While in transit, the encrypted information is unintelligible to an intruder.
- *Tamper detection* allows the recipient of information to verify that it has not been modified in transit. Any attempts to modify or substitute data are detected.
- *Authentication* allows the recipient of information to determine its origin by confirming the sender's identity.
- *Nonrepudiation* prevents the sender of information from claiming at a later date that the information was never sent.

## 1.1. Encryption and Decryption

*Encryption* is the process of transforming information so it is unintelligible to anyone but the intended recipient. *Decryption* is the process of decoding encrypted information. A cryptographic algorithm, also called a *cipher*, is a mathematical function used for encryption or decryption. Usually, two related functions are used, one for encryption and the other for decryption.

With most modern cryptography, the ability to keep encrypted information secret is based not on the cryptographic algorithm, which is widely known, but on a number called a *key* that must be used with the algorithm to produce an encrypted result or to decrypt previously encrypted information. Decryption with the correct key is simple. Decryption without the correct key is very difficult, if not impossible.

### 1.1.1. Symmetric-Key Encryption

With symmetric-key encryption, the encryption key can be calculated from the decryption key and vice versa. With most symmetric algorithms, the same key is used for both encryption and decryption, as shown in Figure 1.1, “Symmetric-Key Encryption”.

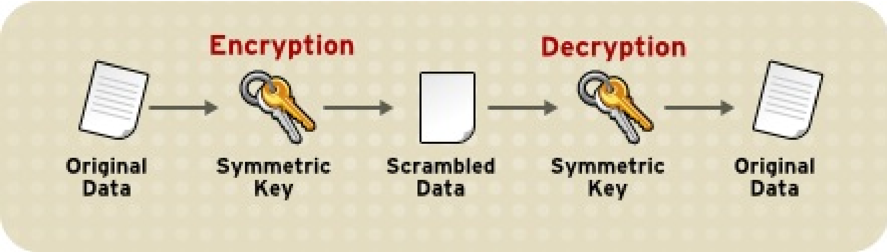



**Figure 1.1. Symmetric-Key Encryption**

Implementations of symmetric-key encryption can be highly efficient, so that users do not experience any significant time delay as a result of the encryption and decryption. Symmetric-key encryption also provides a degree of authentication, since information encrypted with one symmetric key cannot be decrypted with any other symmetric key. Thus, as long as the symmetric key is kept secret by the two parties using it to encrypt communications, each party can be sure that it is communicating with the other as long as the decrypted messages continue to make sense.

Symmetric-key encryption is effective only if the symmetric key is kept secret by the two parties involved. If anyone else discovers the key, it affects both confidentiality and authentication. A person with an unauthorized symmetric key not only can decrypt messages sent with that key, but can encrypt new messages and send them as if they came from one of the legitimate parties using the key.

Symmetric-key encryption plays an important role in SSL communication, which is widely used for authentication, tamper detection, and encryption over TCP/IP networks. SSL also uses techniques of public-key encryption, which is described in the next section.

### 1.1.2. Public-Key Encryption

Public-key encryption (also called asymmetric encryption) involves a pair of keys, a public key and a private key, associated with an entity. Each public key is published, and the corresponding private key is kept secret. (For more information about the way public keys are published, see Section 1.3, “Certificates and Authentication”.) Data encrypted with a public key can be decrypted only with the corresponding private key. Figure 1.2, “Public-Key Encryption” shows a simplified view of the way public-key encryption works.

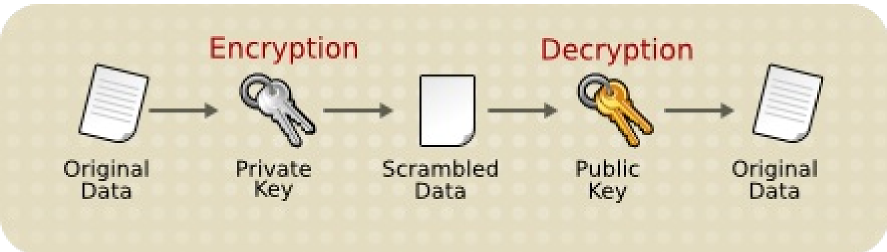



**Figure 1.2. Public-Key Encryption**

The scheme shown in Figure 1.2, “Public-Key Encryption” allows public keys to be freely distributed, while only authorized people are able to read data encrypted using this key. In general, to send encrypted data, the data is encrypted with that person's public key, and the person receiving the encrypted data decrypts it with the corresponding private key.

Compared with symmetric-key encryption, public-key encryption requires more processing and may not be feasible for encrypting and decrypting large amounts of data. However, it is possible to use public-key encryption to send a symmetric key, which can then be used to encrypt additional data. This is the

approach used by the SSL/TLS protocols.

The reverse of the scheme shown in Figure 1.2, “Public-Key Encryption” also works: data encrypted with a private key can be decrypted only with the corresponding public key. This is not a recommended practice to encrypt sensitive data, however, because it means that anyone with the public key, which is by definition published, could decrypt the data. Nevertheless, private-key encryption is useful because it means the private key can be used to sign data with a digital signature, an important requirement for electronic commerce and other commercial applications of cryptography. Client software such as Mozilla Firefox can then use the public key to confirm that the message was signed with the appropriate private key and that it has not been tampered with since being signed. Section 1.2, “Digital Signatures” illustrates how this confirmation process works.

### 1.1.3. Key Length and Encryption Strength

*Breaking* an encryption algorithm is basically finding the key to the access the encrypted data in plain text. For symmetric algorithms, breaking the algorithm usually means trying to determine the key used to encrypt the text. For a public key algorithm, breaking the algorithm usually means acquiring the shared secret information between two recipients.

One method of breaking a symmetric algorithm is to simply try every key within the full algorithm until the right key is found. For public key algorithms, since half of the key pair is publicly known, the other half (private key) can be derived using published, though complex, mathematical calculations. Manually finding the key to break an algorithm is called a brute force attack.

Breaking an algorithm introduces the risk of intercepting, or even impersonating and fraudulently verifying, private information.

The *key strength* of an algorithm is determined by finding the fastest method to break the algorithm and comparing it to a brute force attack.

For symmetric keys, encryption strength is often described in terms of the size or *length* of the keys used to perform the encryption: longer keys generally provide stronger encryption. Key length is measured in bits. For example, 128-bit keys with the RC4 symmetric-key cipher supported by SSL provide significantly better cryptographic protection than 40-bit keys used with the same cipher. The 128-bit RC4 encryption is $3 \times 10^{26}$ times stronger than 40-bit RC4 encryption.

An encryption key is considered full strength if the best known attack to break the key is no faster than a brute force attempt to test every key possibility.

Different types of algorithms — particularly public key algorithms — may require different key lengths to achieve the same level of encryption strength as a symmetric-key cipher. The RSA cipher can use only a subset of all possible values for a key of a given length, due to the nature of the mathematical problem on which it is based. Other ciphers, such as those used for symmetric-key encryption, can use all possible values for a key of a given length. More possible matching options means more security.

Because it is relatively trivial to break an RSA key, an RSA public-key encryption cipher must have a very long key — at least 1024 bits — to be considered cryptographically strong. On the other hand, symmetric-key ciphers are reckoned to be equivalently strong using a much shorter key length, as little as 80 bits for most algorithms.

## 1.2. Digital Signatures

Tamper detection relies on a mathematical function called a *one-way hash* (also called a *message digest*). A one-way hash is a number of fixed length with the following characteristics:

- The value of the hash is unique for the hashed data. Any change in the data, even deleting or altering a single character, results in a different value.
- The content of the hashed data cannot be deduced from the hash.

As mentioned in Section 1.1.2, "Public-Key Encryption", it is possible to use a private key for encryption and the corresponding public key for decryption. Although not recommended when encrypting sensitive information, it is a crucial part of digitally signing any data. Instead of encrypting the data itself, the signing software creates a one-way hash of the data, then uses the private key to encrypt the hash. The encrypted hash, along with other information such as the hashing algorithm, is known as a digital signature.

Figure 1.3, "Using a Digital Signature to Validate Data Integrity" illustrates the way a digital signature can be used to validate the integrity of signed data.

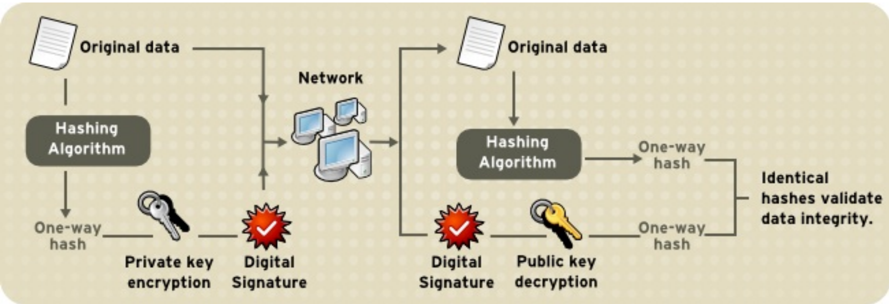



**Figure 1.3. Using a Digital Signature to Validate Data Integrity**

Figure 1.3, "Using a Digital Signature to Validate Data Integrity" shows two items transferred to the recipient of some signed data: the original data and the digital signature, which is a one-way hash of the original data encrypted with the signer's private key. To validate the integrity of the data, the receiving software first uses the public key to decrypt the hash. It then uses the same hashing algorithm that generated the original hash to generate a new one-way hash of the same data. (Information about the hashing algorithm used is sent with the digital signature.) Finally, the receiving software compares the new hash against the original hash. If the two hashes match, the data has not changed since it was signed. If they do not match, the data may have been tampered with since it was signed, or the signature may have been created with a private key that does not correspond to the public key presented by the signer.

If the two hashes match, the recipient can be certain that the public key used to decrypt the digital signature corresponds to the private key used to create the digital signature. Confirming the identity of the signer also requires some way of confirming that the public key belongs to a particular entity. For more information on authenticating users, see Section 1.3, "Certificates and Authentication".

A digital signature is similar to a handwritten signature. Once data have been signed, it is difficult to deny doing so later, assuming the private key has not been compromised. This quality of digital signatures provides a high degree of nonrepudiation; digital signatures make it difficult for the signer to deny having signed the data. In some situations, a digital signature is as legally binding as a handwritten signature.

## 1.3. Certificates and Authentication

- Section 1.3.5, “How CA Certificates Establish Trust”

### 1.3.1. A Certificate Identifies Someone or Something

A certificate is an electronic document used to identify an individual, a server, a company, or other entity and to associate that identity with a public key. Like a driver's license or passport, a certificate provides generally recognized proof of a person's identity. Public-key cryptography uses certificates to address the problem of impersonation.

To get personal ID such as a driver's license, a person has to present some other form of identification which confirms that the person is who he claims to be. Certificates work much the same way. Certificate authorities (CAs) validate identities and issue certificates. CAs can be either independent third parties or organizations running their own certificate-issuing server software, such as Certificate System. The methods used to validate an identity vary depending on the policies of a given CA for the type of certificate being requested. Before issuing a certificate, a CA must confirm the user's identity with its standard verification procedures.

The certificate issued by the CA binds a particular public key to the name of the entity the certificate identifies, such as the name of an employee or a server. Certificates help prevent the use of fake public keys for impersonation. Only the public key certified by the certificate will work with the corresponding private key possessed by the entity identified by the certificate.

In addition to a public key, a certificate always includes the name of the entity it identifies, an expiration date, the name of the CA that issued the certificate, and a serial number. Most importantly, a certificate always includes the digital signature of the issuing CA. The CA's digital signature allows the certificate to serve as a valid credential for users who know and trust the CA but do not know the entity identified by the certificate.

For more information about the role of CAs, see Section 1.3.5, “How CA Certificates Establish Trust”.

### 1.3.2. Authentication Confirms an Identity

*Authentication* is the process of confirming an identity. For network interactions, authentication involves the identification of one party by another party. There are many ways to use authentication over networks. Certificates are one of those way.

Network interactions typically take place between a client, such as a web browser, and a server. *Client authentication* refers to the identification of a client (the person assumed to be using the software) by a server. *Server authentication* refers to the identification of a server (the organization assumed to be running the server at the network address) by a client.

Client and server authentication are not the only forms of authentication that certificates support. For example, the digital signature on an email message, combined with the certificate that identifies the sender, can authenticate the sender of the message. Similarly, a digital signature on an HTML form, combined with a certificate that identifies the signer, can provide evidence that the person identified by that certificate agreed to the contents of the form. In addition to authentication, the digital signature in both cases ensures a degree of nonrepudiation; a digital signature makes it difficult for the signer to claim later not to have sent the email or the form.

Client authentication is an essential element of network security within most intranets or extranets. There are two main forms of client authentication:

- *Password-based authentication* . Almost all server software permits client authentication by requiring a recognized name and password before granting access to the server.
- *Certificate-based authentication* . Client authentication based on certificates is part of the SSL protocol. The client digitally signs a randomly generated piece of data and sends both the certificate

and the signed data across the network. The server validates the signature and confirms the validity of the certificate.

#### 1.3.2.1. Password-Based Authentication

Figure 1.4, “Using a Password to Authenticate a Client to a Server” shows the process of authenticating a user using a username and password. This example assumes the following:

- The user has already trusted the server, either without authentication or on the basis of server authentication over SSL.
- The user has requested a resource controlled by the server.
- The server requires client authentication before permitting access to the requested resource.

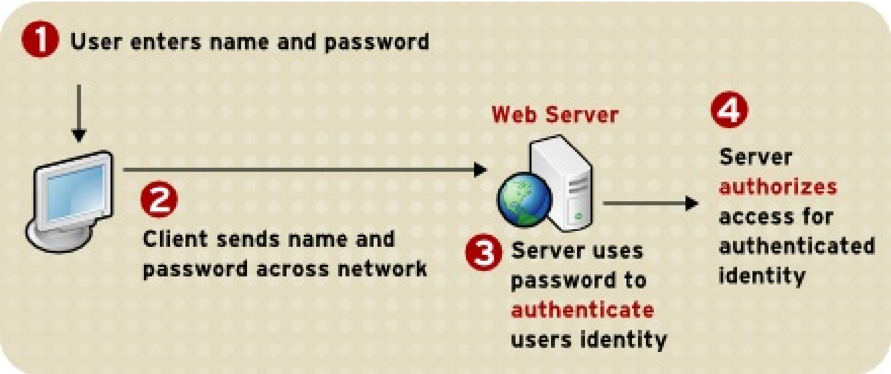



**Figure 1.4. Using a Password to Authenticate a Client to a Server**

These are the steps in this authentication process:

1. When the server requests authentication from the client, the client displays a dialog box requesting the username and password for that server.
2. The client sends the name and password across the network, either in plain text or over an encrypted SSL connection.
3. The server looks up the name and password in its local password database and, if they match, accepts them as evidence authenticating the user's identity.
4. The server determines whether the identified user is permitted to access the requested resource and, if so, allows the client to access it.

With this arrangement, the user must supply a new password for each server accessed, and the administrator must keep track of the name and password for each user.

#### 1.3.2.2. Certificate-Based Authentication

One of the advantages of certificate-based authentication is that it can be used to replace the first three steps in authentication with a mechanism that allows the user to supply one password, which is not sent across the network, and allows the administrator to control user authentication centrally. This is called *single sign-on*.

Figure 1.5, “Using a Certificate to Authenticate a Client to a Server” shows how client authentication works using certificates and SSL. To authenticate a user to a server, a client digitally signs a randomly generated piece of data and sends both the certificate and the signed data across the network. The server authenticates the user's identity based on the data in the certificate and signed data.

Like Figure 1.4, “Using a Password to Authenticate a Client to a Server”, Figure 1.5, “Using a Certificate to Authenticate a Client to a Server” assumes that the user has already trusted the server and

requested a resource and that the server has requested client authentication before granting access to the requested resource.

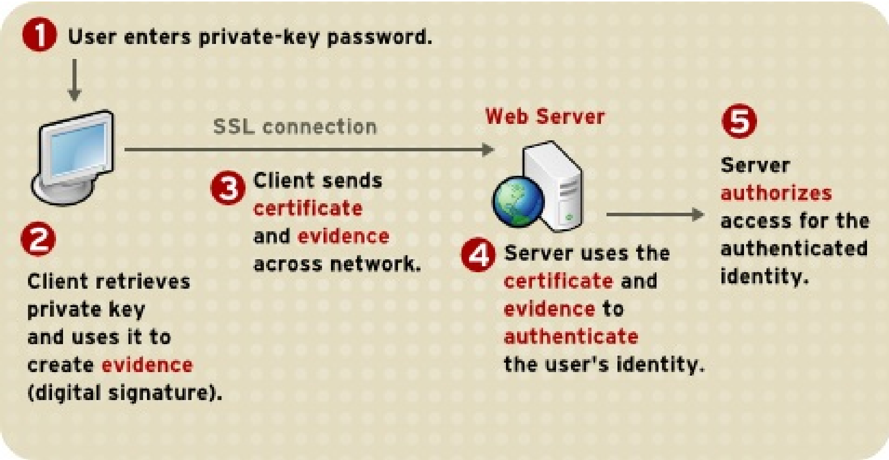



**Figure 1.5. Using a Certificate to Authenticate a Client to a Server**

Unlike the authentication process in Figure 1.4, "Using a Password to Authenticate a Client to a Server", the authentication process in Figure 1.5, "Using a Certificate to Authenticate a Client to a Server" requires SSL. Figure 1.5, "Using a Certificate to Authenticate a Client to a Server" also assumes that the client has a valid certificate that can be used to identify the client to the server. Certificate-based authentication is preferred to password-based authentication because it is based on the user both possessing the private key and knowing the password. However, these two assumptions are true only if unauthorized personnel have not gained access to the user's machine or password, the password for the client software's private key database has been set, and the software is set up to request the password at reasonably frequent intervals.

**NOTE**

Neither password-based authentication nor certificate-based authentication address security issues related to physical access to individual machines or passwords. Public-key cryptography can only verify that a private key used to sign some data corresponds to the public key in a certificate. It is the user's responsibility to protect a machine's physical security and to keep the private-key password secret.

These are the authentication steps shown in Figure 1.5, "Using a Certificate to Authenticate a Client to a Server":

1. The client software maintains a database of the private keys that correspond to the public keys published in any certificates issued for that client. The client asks for the password to this database the first time the client needs to access it during a given session, such as the first time the user attempts to access an SSL-enabled server that requires certificate-based client authentication.

   After entering this password once, the user does not need to enter it again for the rest of the session, even when accessing other SSL-enabled servers.

2. The client unlocks the private-key database, retrieves the private key for the user's certificate, and uses that private key to sign data randomly-generated from input from both the client and the server. This data and the digital signature are evidence of the private key's validity. The digital signature can be created only with that private key and can be validated with the corresponding

public key against the signed data, which is unique to the SSL session.

3. The client sends both the user's certificate and the randomly-generated data across the network.
4. The server uses the certificate and the signed data to authenticate the user's identity.
5. The server may perform other authentication tasks, such as checking that the certificate presented by the client is stored in the user's entry in an LDAP directory. The server then evaluates whether the identified user is permitted to access the requested resource. This evaluation process can employ a variety of standard authorization mechanisms, potentially using additional information in an LDAP directory or company databases. If the result of the evaluation is positive, the server allows the client to access the requested resource.

Certificates replace the authentication portion of the interaction between the client and the server. Instead of requiring a user to send passwords across the network continually, single sign-on requires the user to enter the private-key database password once, without sending it across the network. For the rest of the session, the client presents the user's certificate to authenticate the user to each new server it encounters. Existing authorization mechanisms based on the authenticated user identity are not affected.

## 1.3.3. How Certificates Are Used

Certificates have a purpose: to establish trust. Their usage varies depending on the kind of trust they are used to ensure. Some kinds of certificates are used to verify the identity of the presenter; others are used to verify that an object or item has not been tampered with.

### 1.3.3.1. Uses for Certificates


- Section 1.3.3.1.1, “SSL”
- Section 1.3.3.1.2, “Signed and Encrypted Email”
- Section 1.3.3.1.3, “Single Sign-on”
- Section 1.3.3.1.4, “Object Signing”


#### 1.3.3.1.1. SSL

The Secure Sockets Layer (SSL) protocol governs server authentication, client authentication, and encrypted communication between servers and clients. SSL is widely used on the Internet, especially for interactions that involve exchanging confidential information such as credit card numbers.

SSL requires an SSL server certificate. As part of the initial SSL handshake, the server presents its certificate to the client to authenticate the server's identity. The authentication uses public-key encryption and digital signatures to confirm that the server is the server it claims to be. Once the server has been authenticated, the client and server use symmetric-key encryption, which is very fast, to encrypt all the information exchanged for the remainder of the session and to detect any tampering.

Servers may be configured to require client authentication as well as server authentication. In this case, after server authentication is successfully completed, the client must also present its certificate to the server to authenticate the client's identity before the encrypted SSL session can be established.

For an overview of client authentication over SSL and how it differs from password-based authentication, see Section 1.3.2, “Authentication Confirms an Identity”.

#### 1.3.3.1.2. Signed and Encrypted Email

Some email programs support digitally signed and encrypted email using a widely accepted protocol known as Secure Multipurpose Internet Mail Extension (S/MIME). Using S/MIME to sign or encrypt email messages requires the sender of the message to have an S/MIME certificate.

An email message that includes a digital signature provides some assurance that it was sent by the person whose name appears in the message header, thus authenticating the sender. If the digital signature cannot be validated by the email software, the user is alerted.

The digital signature is unique to the message it accompanies. If the message received differs in any way from the message that was sent, even by adding or deleting a single character, the digital signature cannot be validated. Therefore, signed email also provides assurance that the email has not been tampered with. This kind of assurance is known as nonrepudiation, which makes it difficult for the sender to deny having sent the message. This is important for business communication. For information about the way digital signatures work, see Section 1.2, “Digital Signatures”.

S/MIME also makes it possible to encrypt email messages, which is important for some business users. However, using encryption for email requires careful planning. If the recipient of encrypted email messages loses the private key and does not have access to a backup copy of the key, the encrypted messages can never be decrypted.

#### 1.3.3.1.3. Single Sign-on

Network users are frequently required to remember multiple passwords for the various services they use. For example, a user might have to type a different password to log into the network, collect email, use directory services, use the corporate calendar program, and access various servers. Multiple passwords are an ongoing headache for both users and system administrators. Users have difficulty keeping track of different passwords, tend to choose poor ones, and tend to write them down in obvious places. Administrators must keep track of a separate password database on each server and deal with potential security problems related to the fact that passwords are sent over the network routinely and frequently.

Solving this problem requires some way for a user to log in once, using a single password, and get authenticated access to all network resources that user is authorized to use-without sending any passwords over the network. This capability is known as single sign-on.

Both client SSL certificates and S/MIME certificates can play a significant role in a comprehensive single sign-on solution. For example, one form of single sign-on supported by Red Hat products relies on SSL client authentication. A user can log in once, using a single password to the local client's private-key database, and get authenticated access to all SSL-enabled servers that user is authorized to use-without sending any passwords over the network. This approach simplifies access for users, because they don't need to enter passwords for each new server. It also simplifies network management, since administrators can control access by controlling lists of certificate authorities (CAs) rather than much longer lists of users and passwords.

In addition to using certificates, a complete single-sign on solution must address the need to interoperate with enterprise systems, such as the underlying operating system, that rely on passwords or other forms of authentication.

#### 1.3.3.1.4. Object Signing

Many software technologies support a set of tools called *object signing*. Object signing uses standard techniques of public-key cryptography to let users get reliable information about code they download in much the same way they can get reliable information about shrink-wrapped software.

Most important, object signing helps users and network administrators implement decisions about software distributed over intranets or the Internet-for example, whether to allow Java applets signed by a given entity to use specific computer capabilities on specific users' machines.

The objects signed with object signing technology can be applets or other Java code, JavaScript scripts, plug-ins, or any kind of file. The signature is a digital signature. Signed objects and their signatures are

typically stored in a special file called a JAR file.

Software developers and others who wish to sign files using object-signing technology must first obtain an object-signing certificate.

#### 1.3.3.2. Types of Certificates

The Certificate System is capable of generating different types of certificates for different uses and in different formats. Planning which certificates are required and planning how to manage them, including determining what formats are needed and how to plan for renewal, are important to manage both the PKI and the Certificate System instances.

This list is not exhaustive; there are certificate enrollment forms for dual-use certificates for LDAP directories, file-signing certificates, and other subsystem certificates. These forms are available through the Certificate Manager's end-entities page, at **`https://server.example.com:9444/ca/ee/ca`**. For more detailed information about the different certificates that can be created, see the *Certificate System Agent's Guide*.

When the different Certificate System subsystems are installed, the basic required certificates and keys are generated; for example, configuring the Certificate Manager generates the CA signing certificate for the self-signed root CA and the internal OCSP signing, audit signing, SSL server, and agent user certificates. During the DRM configuration, the Certificate Manager generates the storage, transport, audit signing, and agent certificates. Additional certificates can be created and installed separately.

**Table 1.1. Common Certificates**

| Certificate Type | Use | Example |
|---|---|---|
| Client SSL certificates | Used for client authentication to servers over SSL. Typically, the identity of the client is assumed to be the same as the identity of a person, such as an employee. See Section 1.3.2.2, "Certificate-Based Authentication" for a description of the way SSL client certificates are used for client authentication. Client SSL certificates can also be used as part of single sign-on. | A bank gives a customer an SSL client certificate that allows the bank's servers to identify that customer and authorize access to the customer's accounts.<br>A company gives a new employee an SSL client certificate that allows the company's servers to identify that employee and authorize access to the company's servers. |
| Server SSL certificates | Used for server authentication to clients over SSL. Server authentication may be used without client authentication. Server authentication is required for an encrypted SSL session. For more information, see Section 1.3.3.1.1, "SSL". | Internet sites that engage in electronic commerce usually support certificate-based server authentication to establish an encrypted SSL session and to assure customers that they are dealing with the web site identified with the company. The encrypted SSL session ensures that personal information sent over the network, such as credit card numbers, cannot easily be intercepted. |
| S/MIME certificates | Used for signed and encrypted email. As with SSL client certificates, the identity of the client is assumed to be the same as the identity of a person, such as an employee. A single certificate may be used as both an S/MIME certificate and an SSL certificate; see Section 1.3.3.1.2, "Signed and Encrypted Email". S/MIME certificates can also be used as part of single sign-on. | A company deploys combined S/MIME and SSL certificates solely to authenticate employee identities, thus permitting signed email and SSL client authentication but not encrypted email. Another company issues S/MIME certificates solely to sign and encrypt email that deals with sensitive financial or legal matters. |
| CA certificates | Used to identify CAs. Client and server software use CA certificates to determine what other certificates can be trusted. For more information, see Section 1.3.5, "How CA Certificates Establish Trust". | The CA certificates stored in Mozilla Firefox determine what other certificates can be authenticated. An administrator can implement corporate security policies by controlling the CA certificates stored in each user's copy of Firefox. |
| Object-signing certificates | Used to identify signers of Java code, JavaScript scripts, or other signed files. | Software companies frequently sign software distributed over the Internet to provide users with some assurance that the software is a |

| | |
|---|---|
| | legitimate product of that company. Using certificates and digital signatures can also make it possible for users to identify and control the kind of access downloaded software has to their computers. |



#### 1.3.3.2.1. CA Signing Certificates

Every Certificate Manager has a CA signing certificate with a public/private key pair it uses to sign the certificates and CRLs it issues. This certificate is created and installed when the Certificate Manager is installed.

The Certificate Manager's status as a root or subordinate CA is determined by whether its CA signing certificate is self-signed or is signed by another CA. Self-signed root CAs set the policies they use to issue certificates, such as the subject names, types of certificates that can be issued, and to whom certificates can be issued. A subordinate CA has a CA signing certificate signed by another CA, usually the one that is a level above in the CA hierarchy (which may or may not be a root CA). If the Certificate Manager is a subordinate CA in a CA hierarchy, the root CA's signing certificate must be imported into individual clients and servers before the Certificate Manager can be used to issue certificates to them.

The CA certificate must be installed in a client if a server or user certificate issued by that CA is installed on that client. The CA certificate confirms that the server certificate can be trusted. Ideally, the certificate chain is installed.

#### 1.3.3.2.2. Other Signing Certificates

Other services, such as the OCSP responder service and CRL publishing, can use signing certificates other than the CA certificate. For example, a separate CRL signing certificate can be used to sign the revocation lists that are published by a CA instead of using the CA signing certificate.

#### 1.3.3.2.3. SSL Server and Client Certificates

Server certificates are used for secure communications, such as SSL, and other secure functions. Server certificates are used to authenticate themselves during operations and to encrypt data; client certificates authenticate the client to the server.

> **NOTE**
>
> CAs which have a signing certificate issued by a third-party may not be able to issue server certificates. The third-party CA may have rules in place which prohibit its subordinates from issuing server certificates.

#### 1.3.3.2.4. User Certificates

End user certificates are a subset of client certificates that are used to identify users to a server or system. Users can be assigned certificates to use for secure communications, such as SSL, and other

functions such as encrypting email or for single sign-on. Special users, such as Certificate System agents, can be given client certificates to access special services.

#### 1.3.3.2.5. Dual-Key Pairs

Dual-key pairs are a set of two private and public keys, where one set is used for signing and one for encryption. These dual keys are used to create dual certificates. The dual certificate enrollment form is one of the standard forms listed in the end-entities page of the Certificate Manager.

When generating dual-key pairs, set the certificate profiles to work correctly when generating separate certificates for signing and encryption.

#### 1.3.3.2.6. Cross-Pair Certificates

The Certificate System can issue, import, and publish cross-pair CA certificates. With cross-pair certificates, one CA signs and issues a cross-pair certificate to a second CA, and the second CA signs and issues a cross-pair certificate to the first CA. Both CAs then store or publish both certificates as a **`crossCertificatePair`** entry.

Bridging certificates can be done to honor certificates issued by a CA that is not chained to the root CA. By establishing a trust between the Certificate System CA and another CA through a cross-pair CA certificate, the cross-pair certificate can be downloaded and used to trust the certificates issued by the other CA.

## 1.3.4. Contents of a Certificate

The contents of certificates are organized according to the X.509 v3 certificate specification, which has been recommended by the International Telecommunications Union (ITU), an international standards body.

Users do not usually need to be concerned about the exact contents of a certificate. However, system administrators working with certificates may need some familiarity with the information contained in them.

### 1.3.4.1. Certificate Data Formats

Certificate requests and certificates can be created, stored, and installed in several different formats. All of these formats conform to X.509 standards.

#### 1.3.4.1.1. Binary

The following binary formats are recognized:

- *DER-encoded certificate*. This is a single binary DER-encoded certificate.
- *PKCS #7 certificate chain*. This is a PKCS #7 **`SignedData`** object. The only significant field in the **`SignedData`** object is the certificates; the signature and the contents, for example, are ignored. The PKCS #7 format allows multiple certificates to be downloaded at a single time.
- *Netscape Certificate Sequence*. This is a simpler format for downloading certificate chains in a PKCS #7 **`ContentInfo`** structure, wrapping a sequence of certificates. The value of the **`contentType`** field should be **`netscape-cert-sequence`**, while the content field has the following structure:

  ```
  CertificateSequence ::= SEQUENCE OF Certificate
  ```

  This format allows multiple certificates to be downloaded at the same time.

#### 1.3.4.1.2. Text

Any of the binary formats can be imported in text form. The text form begins with the following line:

```
-----BEGIN CERTIFICATE-----
```

Following this line is the certificate data, which can be in any of the binary formats described. This data should be base-64 encoded, as described by RFC 1113. The certificate information is followed by this line:

```
-----END CERTIFICATE-----
```

### 1.3.4.2. Distinguished Names

An X.509 v3 certificate binds a distinguished name (DN) to a public key. A DN is a series of name-value pairs, such as **`uid=doe`**, that uniquely identify an entity. This is also called the certificate *subject name*.

This is an example DN of an employee for Example Corp.:

```
uid=doe, cn=John Doe,o=Example Corp.,c=US
```

In this DN, **`uid`** is the username, **`cn`** is the user's common name, **`o`** is the organization or company name, and **`c`** is the country.

DNs may include a variety of other name-value pairs. They are used to identify both certificate subjects and entries in directories that support the Lightweight Directory Access Protocol (LDAP).

The rules governing the construction of DNs can be complex; for comprehensive information about DNs, see *A String Representation of Distinguished Names* at http://www.ietf.org/rfc/rfc4514.txt.

### 1.3.4.3. A Typical Certificate

Every X.509 certificate consists of two sections:

- The data section includes the following information:
  - The version number of the X.509 standard supported by the certificate.
  - The certificate's serial number. Every certificate issued by a CA has a serial number that is unique among the certificates issued by that CA.
  - Information about the user's public key, including the algorithm used and a representation of the key itself.
  - The DN of the CA that issued the certificate.
  - The period during which the certificate is valid; for example, between 1:00 p.m. on November 15, 2004, and 1:00 p.m. November 15, 2009.
  - The DN of the certificate subject, which is also called the subject name; for example, in an SSL client certificate, this is the user's DN.
  - Optional *certificate extensions*, which may provide additional data used by the client or server. For example, the Netscape Certificate Type extension indicates the type of certificate, such as an SSL client certificate, an SSL server certificate, or a certificate for signing email. Certificate extensions can also be used for other purposes.
- The signature section includes the following information:
  - The cryptographic algorithm, or cipher, used by the issuing CA to create its own digital signature.
  - The CA's digital signature, obtained by hashing all of the data in the certificate together and encrypting it with the CA's private key.

Here are the data and signature sections of a certificate shown in the readable pretty-print format:

```
Certificate:
Data:
   Version: v3 (0x2)
   Serial Number: 3 (0x3)
   Signature Algorithm: PKCS #1 MD5 With RSA Encryption
   Issuer: OU=Example Certificate Authority, O=Example Corp, C=US
   Validity:
    Not Before: Fri Oct 17 18:36:25 1997
    Not  After: Sun Oct 17 18:36:25 1999
   Subject: CN=Jane Doe, OU=Finance, O=Example Corp, C=US
   Subject Public Key Info:
    Algorithm: PKCS #1 RSA Encryption
    Public Key:
       Modulus:
          00:ca:fa:79:98:8f:19:f8:d7:de:e4:49:80:48:e6:2a:2a:86:
          ed:27:40:4d:86:b3:05:c0:01:bb:50:15:c9:de:dc:85:19:22:
          43:7d:45:6d:71:4e:17:3d:f0:36:4b:5b:7f:a8:51:a3:a1:00:
          98:ce:7f:47:50:2c:93:36:7c:01:6e:cb:89:06:41:72:b5:e9:
          73:49:38:76:ef:b6:8f:ac:49:bb:63:0f:9b:ff:16:2a:e3:0e:
          9d:3b:af:ce:9a:3e:48:65:de:96:61:d5:0a:11:2a:a2:80:b0:
          7d:d8:99:cb:0c:99:34:c9:ab:25:06:a8:31:ad:8c:4b:aa:54:
          91:f4:15
       Public Exponent: 65537 (0x10001)
   Extensions:
    Identifier: Certificate Type
      Critical: no
      Certified Usage:
      SSL Client
    Identifier: Authority Key Identifier
      Critical: no
      Key Identifier:
        f2:f2:06:59:90:18:47:51:f5:89:33:5a:31:7a:e6:5c:fb:36:
        26:c9
   Signature:
    Algorithm: PKCS #1 MD5 With RSA Encryption
   Signature:
 6d:23:af:f3:d3:b6:7a:df:90:df:cd:7e:18:6c:01:69:8e:54:65:fc:06:
 30:43:34:d1:63:1f:06:7d:c3:40:a8:2a:82:c1:a4:83:2a:fb:2e:8f:fb:
 f0:6d:ff:75:a3:78:f7:52:47:46:62:97:1d:d9:c6:11:0a:02:a2:e0:cc:
 2a:75:6c:8b:b6:9b:87:00:7d:7c:84:76:79:ba:f8:b4:d2:62:58:c3:c5:
 b6:c1:43:ac:63:44:42:fd:af:c8:0f:2f:38:85:6d:d6:59:e8:41:42:a5:
 4a:e5:26:38:ff:32:78:a1:38:f1:ed:dc:0d:31:d1:b0:6d:67:e9:46:a8:
 d:c4
```

Here is the same certificate in the base-64 encoded format:

```
-----BEGIN CERTIFICATE-----
MIICKzCCAZSgAwIBAgIBAzANBgkqhkiG9w0BAQQFADA3MQswCQYDVQQGEwJVUzER
MA8GA1UEChMITmV0c2NhcGUxFTATBgNVBAsTDFN1cHJpeWEncyBDQTAeFw05NzEw
MTgwMTM2MjVaFw05OTEwMTgwMTM2MjVaMEgxCzAJBgNVBAYTAlVTMREwDwYDVQQK
EwhOZXRzY2FwZTENMAsGA1UECxMEUHViczEXMBUGA1UEAxMOU3Vwcml5YSBTaGV0
dHkwgZ8wDQYJKoZIhvcNAQEFBQADgY0AMIGJAoGBAMr6eZiPGfjX3uRJgEjmKiqG
7SdATYazBcABu1AVyd7chRkiQ31FbXFOGD3wNktbf6hRo6EAmM5/R1AskzZ8AW7L
iQZBcrXpc0k4du+2Q6xJu2MPm/8WKuMOnTuvzpo+SGXelmHVChEqooCwfdiZywyZ
NMmrJgaoMa2MS6pUkfQVAgMBAAGjNjA0MBEGCWCGSAGG+EIBAQQEAwIAgDAfBgNV
HSMEGDAWgBTy8gZZkBhHUfWJM1oxeuZc+zYmyTANBgkqhkiG9w0BAQQFAAOBgQBt
I6/z07Z635DfzX4XbAFpjlRl/AYwQzTSYx8GfcNAqCqCwaSDKvsuj/vwbf91o3j3
UkdGYpcd2cYRCgKi4MwqdWyLtpuHAH18hHZ5uvi00mJYw8W2wUOsY0RC/a/IDy84
hW3WWehBUqVK5SY4/zJ4oTjx7dwNMdGwbWfpRqjd1A==
-----END CERTIFICATE-----
```


## 1.3.5. How CA Certificates Establish Trust

CAs validate identities and issue certificates. They can be either independent third parties or organizations running their own certificate-issuing server software, such as the Certificate System.

Any client or server software that supports certificates maintains a collection of trusted CA certificates. These CA certificates determine which issuers of certificates the software can trust, or validate. In the simplest case, the software can validate only certificates issued by one of the CAs for which it has a certificate. It is also possible for a trusted CA certificate to be part of a chain of CA certificates, each issued by the CA above it in a certificate hierarchy.

The sections that follow explains how certificate hierarchies and certificate chains determine what certificates software can trust.


- Section 1.3.5.1, “CA Hierarchies”
- Section 1.3.5.2, “Certificate Chains”
- Section 1.3.5.3, “Verifying a Certificate Chain”


### 1.3.5.1. CA Hierarchies

In large organizations, responsibility for issuing certificates can be delegated to several different CAs. For example, the number of certificates required may be too large for a single CA to maintain; different organizational units may have different policy requirements; or a CA may need to be physically located in the same geographic area as the people to whom it is issuing certificates.

These certificate-issuing responsibilities can be divided among subordinate CAs. The X.509 standard includes a model for setting up a hierarchy of CAs, shown in Figure 1.6, “Example of a Hierarchy of Certificate Authorities”.

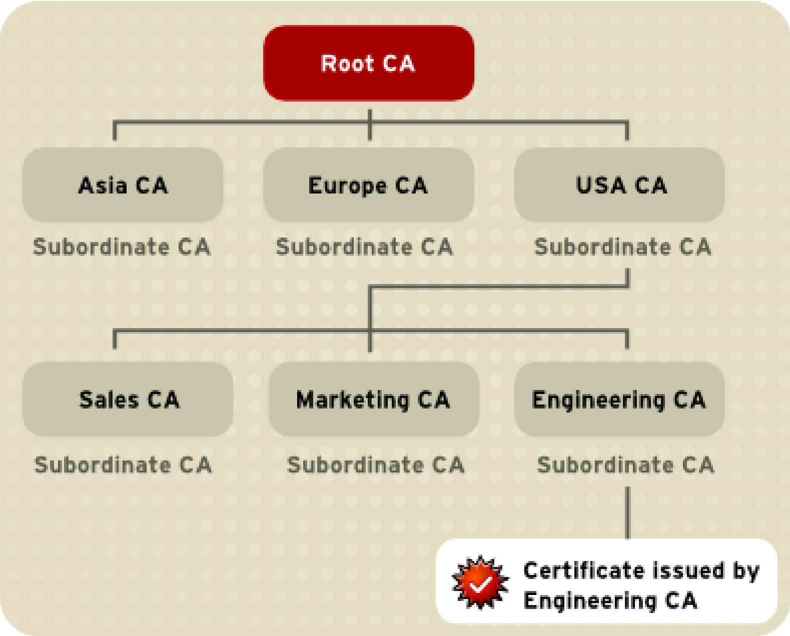



**Figure 1.6. Example of a Hierarchy of Certificate Authorities**

The root CA is at the top of the hierarchy. The root CA's certificate is a *self-signed certificate*; that is, the certificate is digitally signed by the same entity that the certificate identifies. The CAs that are directly subordinate to the root CA have CA certificates signed by the root CA. CAs under the subordinate CAs in the hierarchy have their CA certificates signed by the higher-level subordinate CAs.

Organizations have a great deal of flexibility in how CA hierarchies are set up; Figure 1.6, “Example of a Hierarchy of Certificate Authorities” shows just one example.

### 1.3.5.2. Certificate Chains

CA hierarchies are reflected in certificate chains. A *certificate chain* is series of certificates issued by successive CAs. Figure 1.7, “Example of a Certificate Chain” shows a certificate chain leading from a certificate that identifies an entity through two subordinate CA certificates to the CA certificate for the root CA, based on the CA hierarchy shown in Figure 1.6, “Example of a Hierarchy of Certificate Authorities”.

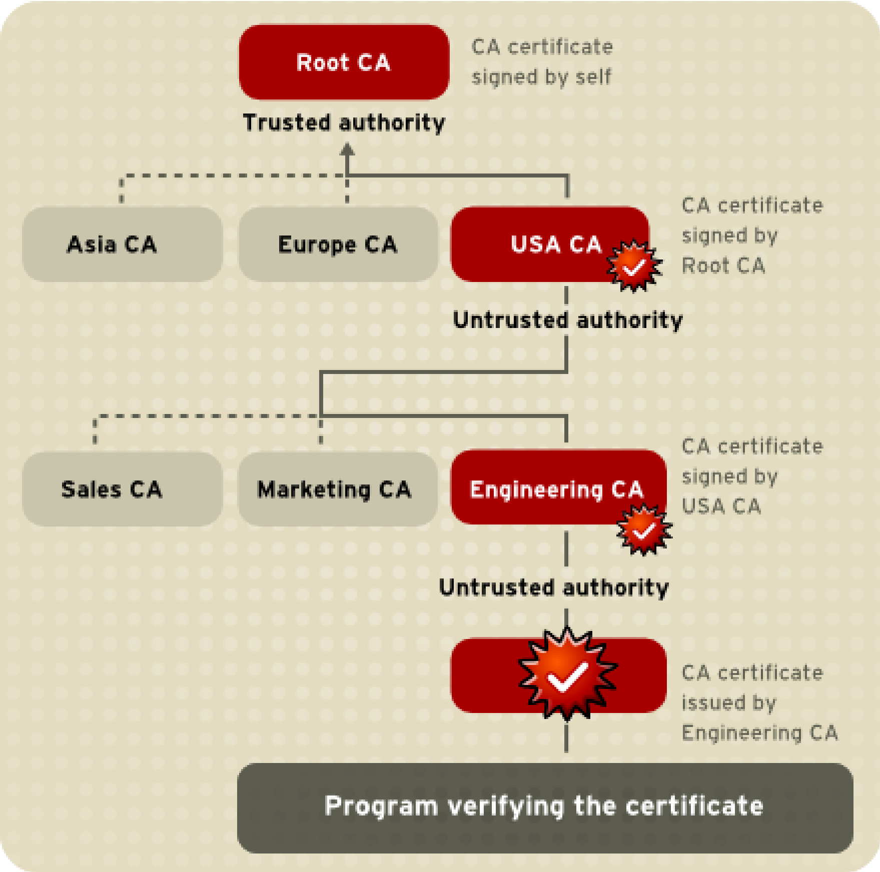



**Figure 1.7. Example of a Certificate Chain**

A certificate chain traces a path of certificates from a branch in the hierarchy to the root of the hierarchy. In a certificate chain, the following occur:

- Each certificate is followed by the certificate of its issuer.
- Each certificate contains the name (DN) of that certificate's issuer, which is the same as the subject name of the next certificate in the chain.

  In Figure 1.7, “Example of a Certificate Chain”, the **`Engineering CA`** certificate contains the DN of the CA, **`USA CA`**, that issued that certificate. **`USA CA`**'s DN is also the subject name of the next certificate in the chain.
- Each certificate is signed with the private key of its issuer. The signature can be verified with the public key in the issuer's certificate, which is the next certificate in the chain.

  In Figure 1.7, “Example of a Certificate Chain”, the public key in the certificate for the **`USA CA`** can be used to verify the **`USA CA`**'s digital signature on the certificate for the **`Engineering CA`**.

### 1.3.5.3. Verifying a Certificate Chain

Certificate chain verification makes sure a given certificate chain is well-formed, valid, properly signed, and trustworthy. The following procedure is used to form and verify a certificate chain, starting with the certificate being presented for authentication:

1. The certificate validity period is checked against the current time provided by the verifier's system clock.
2. The issuer's certificate is located. The source can be either the verifier's local certificate database on that client or server or the certificate chain provided by the subject, as with an SSL connection.
3. The certificate signature is verified using the public key in the issuer's certificate.

4. If the issuer's certificate is trusted by the verifier in the verifier's certificate database, verification stops successfully here. Otherwise, the issuer's certificate is checked to make sure it contains the appropriate subordinate CA indication in the certificate type extension, and chain verification starts over with this new certificate. Figure 1.8, “Verifying a Certificate Chain to the Root CA” presents an example of this process.

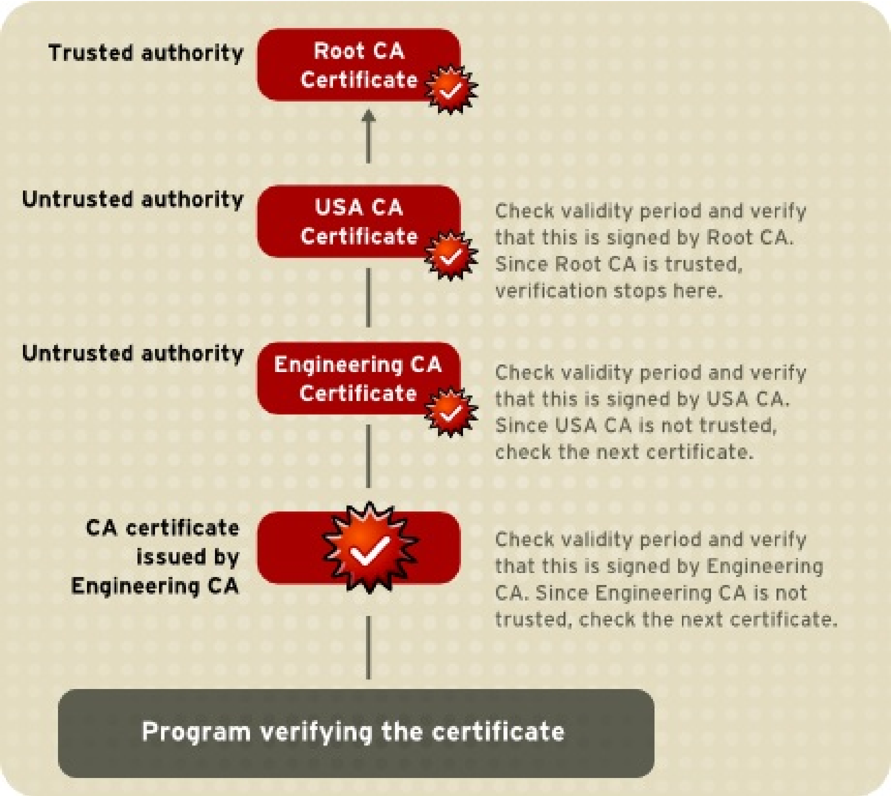



**Figure 1.8. Verifying a Certificate Chain to the Root CA**

Figure 1.8, “Verifying a Certificate Chain to the Root CA” illustrates what happens when only the root CA is included in the verifier's local database. If a certificate for one of the intermediate CAs, such as `Engineering CA`, is found in the verifier's local database, verification stops with that certificate, as shown in Figure 1.9, “Verifying a Certificate Chain to an Intermediate CA”.

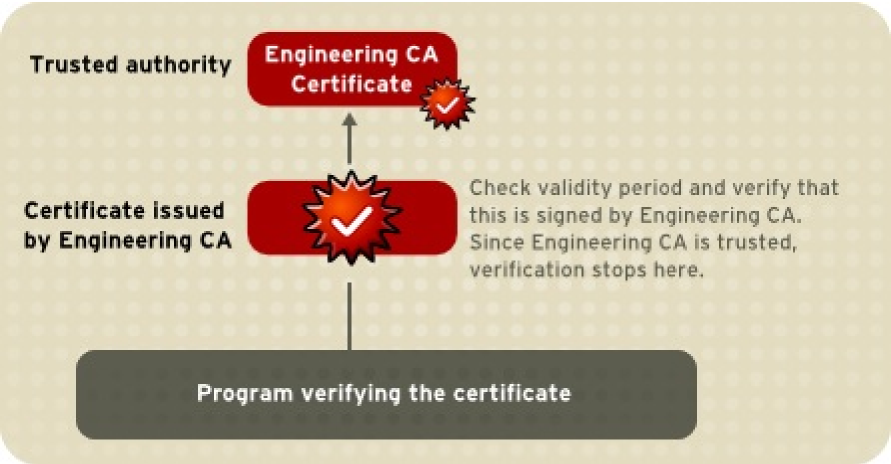



**Figure 1.9. Verifying a Certificate Chain to an Intermediate CA**

Expired validity dates, an invalid signature, or the absence of a certificate for the issuing CA at any point in the certificate chain causes authentication to fail. Figure 1.10, “A Certificate Chain That Cannot Be Verified” shows how verification fails if neither the root CA certificate nor any of the intermediate CA certificates are included in the verifier's local database.

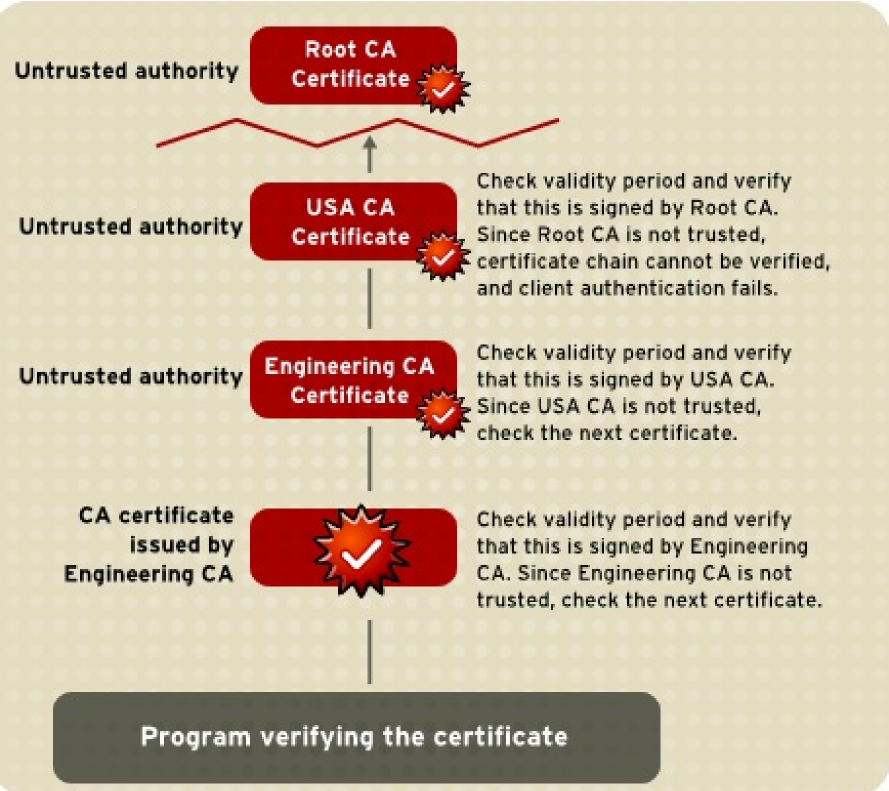



**Figure 1.10. A Certificate Chain That Cannot Be Verified**

## 1.4. Managing Certificates

Certificates are used in many applications, from encrypting email to accessing websites. There are two major stages in the lifecycle of the certificate: the point when it is issued (issuance and enrollment) and the period when the certificates are no longer valid (renewal or revocation). There are also ways to manage the certificate during its cycle. Making information about the certificate available to other applications is *publishing* the certificate and then backing up the key pairs so that the certificate can be recovered if it is lost.

- Section 1.4.1, “Issuing Certificates”
- Section 1.4.2, “Key Management”
- Section 1.4.3, “Renewing and Revoking Certificates”

### 1.4.1. Issuing Certificates

The process for issuing a certificate depends on the CA that issues it and the purpose for which it will be used. Issuing non-digital forms of identification varies in similar ways. The requirements to get a library card are different than the ones to get a driver's license. Similarly, different CAs have different procedures for issuing different kinds of certificates. Requirements for receiving a certificate can be as simple as an email address or username and password to notarized documents, a background check, and a personal interview.

Depending on an organization's policies, the process of issuing certificates can range from being completely transparent for the user to requiring significant user participation and complex procedures. In general, processes for issuing certificates should be flexible, so organizations can tailor them to their changing needs.

### 1.4.2. Key Management

Before a certificate can be issued, the public key it contains and the corresponding private key must be generated. Sometimes it may be useful to issue a single person one certificate and key pair for signing operations and another certificate and key pair for encryption operations. Separate signing and encryption certificates keep the private signing key only on the local machine, providing maximum nonrepudiation. This also aids in backing up the private encryption key in some central location where it can be retrieved in case the user loses the original key or leaves the company.

Keys can be generated by client software or generated centrally by the CA and distributed to users through an LDAP directory. There are costs associated with either method. Local key generation provides maximum nonrepudiation but may involve more participation by the user in the issuing process. Flexible key management capabilities are essential for most organizations.

*Key recovery* , or the ability to retrieve backups of encryption keys under carefully defined conditions, can be a crucial part of certificate management, depending on how an organization uses certificates. In some PKI setups, several authorized personnel must agree before an encryption key can be recovered to ensure that the key is only recovered to the legitimate owner in authorized circumstance. It can be necessary to recover a key when information is encrypted and can only be decrypted by the lost key.

### 1.4.3. Renewing and Revoking Certificates

Like a driver's license, a certificate specifies a period of time during which it is valid. Attempts to use a certificate for authentication before or after its validity period will fail. Managing certificate expirations and renewals are an essential part of the certificate management strategy. For example, an administrator may wish to be notified automatically when a certificate is about to expire so that an appropriate renewal process can be completed without disrupting the system operation. The renewal process may involve reusing the same public-private key pair or issuing a new one.

Additionally, it may be necessary to revoke a certificate before it has expired, such as when an employee leaves a company or moves to a new job in a different unit within the company.

Certificate revocation can be handled in several different ways. Servers can be configured so that the authentication process checks the directory for the presence of the certificate being presented. When an administrator revokes a certificate, the certificate can be automatically removed from the directory, and subsequent authentication attempts with that certificate will fail, even though the certificate remains valid in every other respect. Alternatively, a list of revoked certificates, a certificate revocation list (CRL), can be published to the directory at regular intervals. The CRL can be checked as part of the authentication process. The issuing CA can also be checked directly each time a certificate is presented for authentication. This procedure is sometimes called real-time status checking.

# Chapter 2. Overview of Red Hat Certificate System Subsystems

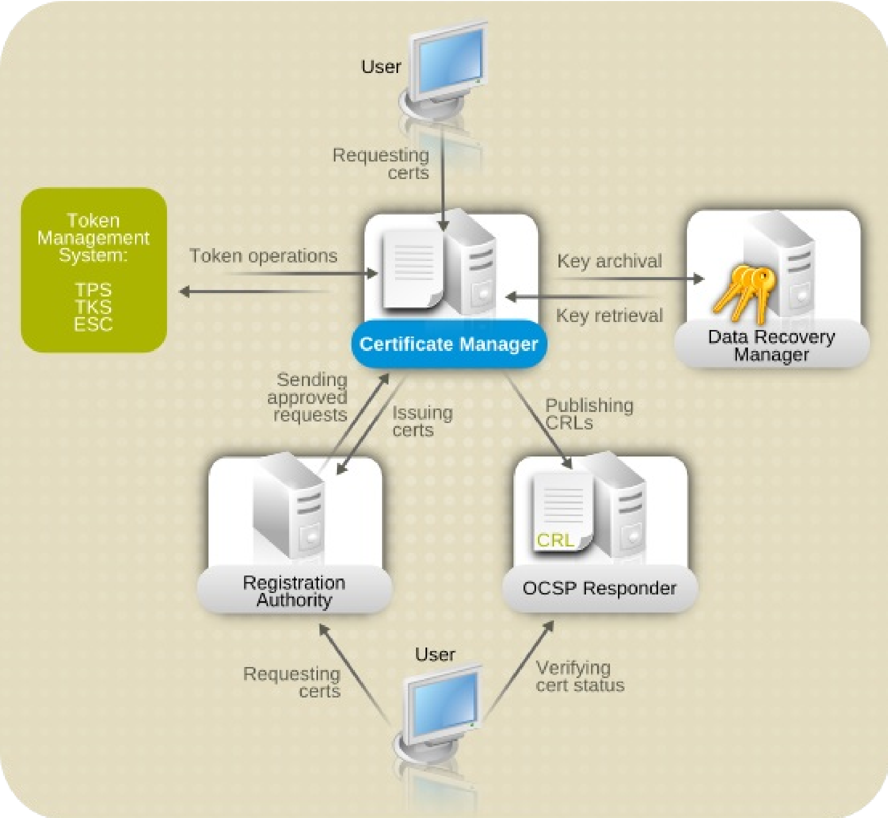
Every common PKI operation — issuing, renewing and revoking certificates; archiving and recovering keys; publishing CRLs and verifying certificate status — are carried out by interoperating subsystems within Red Hat Certificate System. The functions of each individual subsystem and the way that they work together to establish a robust and local PKI is described in this chapter.

## 2.1. A Review of Certificate System Subsystems

Red Hat Certificate System provides six different subsystems, each focusing on different aspects of a PKI deployment:

- A *certificate authority* called a *Certificate Manager*. The CA is the core of the PKI; it issues and revokes all certificates. The Certificate Manager is also the core of the Certificate System. By establishing a *security domain* of trusted subsystems, it establishes and manages relationships between the other subsystems.
- A *key recovery authority* called a *data recovery manager* (DRM). Certificates are created based on a specific and unique key pair. If a private key is ever lost, then the data which that key was used to access (such as encrypted emails) is also lost because it is inaccessible. The DRM stores key pairs, so that a new, identical certificate can be generated based on recovered keys, and all of the encrypted data can be accessed even after a private key is lost or damaged.
- An *online certificate status responder* (OCSP). The OCSP verifies whether a certificate is valid and not expired. This function can also be done by the CA, which has an internal OCSP service, but using an external OCSP eases the load off of the issuing CA.
- A *registration authority* (RA). An RA accepts certificate requests and verifies, independently, whether that request should be approved. It then forwards approved requests to the CA to issue the certificate. Like the OCSP, this is a function that can be performed by the CA, but using a separate subsystem reduces the load on the CA.
- A *token key service* (TKS). The TKS derives keys based on the token CCID, private information, and a defined algorithm. These derived keys are used by the TPS to format tokens and enroll, or process, certificates on the token.
- A *token processing system* (TPS). The TPS interacts directly with external tokens, like smart cards, and manages the keys and certificates on those tokens through a local client, the Enterprise Security Client. The Enterprise Security Client contacts the TPS when there is a token operation, and the TPS interacts with the CA, DRM, or TKS, as required, then send the information back to the token by way of the Enterprise Security Client.

Even with all possible subsystems installed, the core of the Certificate System is still the CA (or CAs), since they ultimately process all certificate-related requests. The other subsystems connect to the CA or CAs likes spokes in a wheel.




## 2.1.1. About the Certificate Manager (CA)

As stated, the Certificate Manager is the heart of the Certificate System. It manages certificates at every stage, from requests through enrollment (issuing), renewal, and revocation. The CA also publishes certificates and lists of revoked certificates for use by clients like the OCSP or web servers.

### 2.1.1.1. Issuing Certificates (Enrollment)

An end entity enrolls in the PKI by submitting an enrollment request through the end-entity interface. There can be many kinds of enrollment that use different enrollment methods or require different authentication methods. For each enrollment, there is a separate enrollment page created that is specific to the type of enrollment, type of authentication, and the certificate profiles associated with the type of certificate. The forms associated with enrollment can be customized for both appearance and content. Alternatively, the enrollment process can be customized by creating certificate profiles for each enrollment type. Certificate profiles dynamically-generate forms which are customized by configuring the inputs associated with the certificate profile.

When an end entity enrolls in a PKI by requesting a certificate, the following events can occur, depending on the configuration of the PKI and the subsystems installed:

1. The end entity provides the information in one of the enrollment forms and submits a request.

   The information gathered from the end entity is customizable in the form depending on the information collected to store in the certificate or to authenticate against the authentication method associated with the form. The form creates a request that is then submitted to the Certificate Manager.
2. The enrollment form triggers the creation of the public and private keys or for dual-key pairs for the request.
3. The end entity provides authentication credentials before submitting the request, depending on the authentication type. This can be LDAP authentication, PIN-based authentication, or certificate-based authentication.

4. The request is submitted either to an agent-approved enrollment process or an automated process.
   - The agent-approved process, which involves no end-entity authentication, sends the request to the request queue in the agent services interface, where an agent must processes the request. An agent can then modify parts of the request, change the status of the request, reject the request, or approve the request.

     Automatic notification can be set up so an email is sent to an agent any time a request appears in the queue. Also, an automated job can be set to send a list of the contents of the queue to agents on a pre configured schedule.
   - The automated process, which involves end-entity authentication, processes the certificate request as soon as the end entity successfully authenticates.
5. The form collects information about the end entity from an LDAP directory when the form is submitted. For certificate profile-based enrollment, the defaults for the form can be used to collect the user LDAP ID and password.
6. The certificate profile associated with the form determine aspects of the certificate that is issued. Depending on the certificate profile, the request is evaluated to determine if the request meets the constraints set, if the required information is provided, and the contents of the new certificate.
7. The form can also request that the user export the private encryption key. If the DRM subsystem is set up with this CA, the end entity's key is requested, and an archival request is sent to the DRM. This process generally requires no interaction from the end entity.
8. The certificate request is either rejected because it did not meet the certificate profile or authentication requirements, or a certificate is issued.
9. The certificate is delivered to the end entity.
   - In automated enrollment, the certificate is delivered to the user immediately. Since the enrollment is normally through an HTML page, the certificate is returned as a response on another HTML page.
   - In agent-approved enrollment, the certificate can be retrieved by serial number or request Id in the end-entity interface.
   - If the notification feature is set up, the link where the certificate can be obtained is sent to the end user.
10. An automatic notice can be sent to the end entity when the certificate is issued or rejected.
11. The new certificate is stored in the Certificate Manager's internal database.
12. If publishing is set up for the Certificate Manager, the certificate is published to a file or an LDAP directory.
13. The internal OCSP service checks the status of certificates in the internal database when a certificate status request is received.

The end-entity interface has a search form for certificates that have been issued and for the CA certificate chain.

#### 2.1.1.2. Renewal

When certificates reach their expiration date, they can either be allowed to lapse, or they can be renewed.

*Renewal* regenerates a certificate request using the existing key pairs for that certificate, and then resubmits the request to Certificate Manager. The renewed certificate is identical to the original (since it was created from the same profile using the same key material) with one exception — it has a different, later expiration date.

Renewal can make managing certificates and relationships between users and servers much smoother,

because the renewed certificate functions precisely as the old one. For user certificates, renewal allows encrypted data to be accessed without any loss.

#### 2.1.1.3. Revocation

End entities can request that their own certificates be revoked. When an end entity makes the request, the certificate has to be presented to the CA. If the certificate and the keys are available, the request is processed and sent to the Certificate Manager, and the certificate is revoked. The Certificate Manager marks the certificate as revoked in its database and adds it to any applicable CRLs.

An agent can revoke any certificate issued by the Certificate Manager by searching for the certificate in the agent services interface and then marking it revoked. Once a certificate is revoked, it is marked revoked in the database and in the publishing directory, if the Certificate is set up for publishing.

If the internal OCSP service has been configured, the service determines the status of certificates by looking them up in the internal database.

Automated notifications can be set to send email messages to end entities when their certificates are revoked by enabling and configuring the certificate revoked notification message.

### 2.1.2. About the Registration Manager (RA)

A registration authority (RA) is an intermediary between a user and a CA. It accepts enrollment requests and then authenticates them locally. If the request is approved, the RA sends the request to the CA to issue the certificate and, once the certificate is issued, sends the certificate back to the user.

RAs remove some of the load from CAs by handling the validation part of a certificate request. For example, offices or organizations can validate requests locally, according to their predefined standards, using RA agents. This requires fewer CAs and allows the organization to group all of the CAs in a separate, secure location.

The kinds of certificates that can generated in the Certificate System RA are limited to the most common types of certificates: user certificates, SSL client certificates, RA agent certificates, and SCEP certificates for local routers. Users can check the RA services pages to view their certificate request status, retrieve their issued certificates, and renew their certificates.

Certificate System RAs can perform either automatic approval and manual approval, depending on the configuration in the enrollment profiles. Like the CA, the RA can also be configured to send a notification when the certificate request is processed.

The RA is normally set up outside of the firewall, and the CA is set up behind the firewall. This enables requests to be made from outside the protected environment (for example, the Internet), while the CA remains under the protection of the site's security measures.

### 2.1.3. About OCSP Services

The Certificate System CA supports the Online Certificate Status Protocol (OCSP) as defined in PKIX standard RFC 2560. The OCSP protocol enables OCSP-compliant applications to determine the state of a certificate, including the revocation status, without having to directly check a CRL published by a CA to the validation authority. The validation authority, which is also called an *OCSP responder*, checks for the application.

1. A CA is set up to issue certificates that include the Authority Information Access extension, which identifies an OCSP responder that can be queried for the status of the certificate.
2. The CA periodically publishes CRLs to an OCSP responder.
3. The OCSP responder maintains the CRL it receives from the CA.

4. An OCSP-compliant client sends requests containing all the information required to identify the certificate to the OCSP responder for verification. The applications determine the location of the OCSP responder from the value of the Authority Information Access extension in the certificate being validated.
5. The OCSP responder determines if the request contains all the information required to process it. If it does not or if it is not enabled for the requested service, a rejection notice is sent. If it does have enough information, it processes the request and sends back a report stating the status of the certificate.

### 2.1.3.1. OCSP Response Signing

Every response that the client receives, including a rejection notification, is digitally signed by the responder; the client is expected to verify the signature to ensure that the response came from the responder to which it submitted the request. The key the responder uses to sign the message depends on how the OCSP responder is deployed in a PKI setup. RFC 2560 recommends that the key used to sign the response belong to one of the following:

- The CA that issued the certificate that's status is being checked.
- A responder with a public key trusted by the client. Such a responder is called a *trusted responder*.
- A responder that holds a specially marked certificate issued to it directly by the CA that revokes the certificates and publishes the CRL. Possession of this certificate by a responder indicates that the CA has authorized the responder to issue OCSP responses for certificates revoked by the CA. Such a responder is called a *CA-designated responder* or a *CA-authorized responder*.

The end-entities page of a Certificate Manager includes a form for manually requesting a certificate for the OCSP responder. The default enrollment form includes all the attributes that identify the certificate as an OCSP responder certificate. The required certificate extensions, such as OCSPNoCheck and Extended Key Usage, can be added to the certificate when the certificate request is submitted.

### 2.1.3.2. OCSP Responses

The OCSP response that the client receives indicates the current status of the certificate as determined by the OCSP responder. The response could be any of the following:

- *Good or Verified* . Specifies a positive response to the status inquiry, meaning the certificate has not been revoked. It does not necessarily mean that the certificate was issued or that it is within the certificate's validity interval. Response extensions may be used to convey additional information on assertions made by the responder regarding the status of the certificate.
- *Revoked* . Specifies that the certificate has been revoked, either permanently or temporarily.

Based on the status, the client decides whether to validate the certificate.

**NOTE**

The OCSP responder will never return a response of *Unknown*. The response will always be either *Good* or *Revoked*.

### 2.1.3.3. OCSP Services

There are two ways to set up OCSP services:

- The OCSP built into the Certificate Manager
- The Online Certificate Status Manager subsystem

In addition to the built-in OCSP service, the Certificate Manager can publish CRLs to an OCSP-compliant validation authority. CAs can be configured to publish CRLs to the Certificate System Online Certificate Status Manager. The Online Certificate Status Manager stores each Certificate Manager's CRL in its internal database and uses the appropriate CRL to verify the revocation status of a certificate when queried by an OCSP-compliant client.

The Certificate Manager can generate and publish CRLs whenever a certificate is revoked and at specified intervals. Because the purpose of an OCSP responder is to facilitate immediate verification of certificates, the Certificate Manager should publish the CRL to the Online Certificate Status Manager every time a certificate is revoked. Publishing only at intervals means that the OCSP service is checking an outdated CRL.

**NOTE**

If the CRL is large, the Certificate Manager can take a considerable amount of time to publish the CRL.

The Online Certificate Status Manager stores each Certificate Manager's CRL in its internal database and uses it as the CRL to verify certificates. The Online Certificate Status Manager can also use the CRL published to an LDAP directory, meaning the Certificate Manager does not have to update the CRLs directly to the Online Certificate Status Manager.

## 2.1.4. About the Data Recovery Manager (DRM)

To archive private encryption keys and recover them later, the PKI configuration must include the following elements:

- Clients that can generate dual keys and that support the key archival option (using the CRMF/CMMF protocol).
- An installed and configured DRM.
- HTML forms with which end entities can request dual certificates (based on dual keys) and key recovery agents can request key recovery.

Only keys that are used exclusively for encrypting data should be archived; signing keys in particular should never be archived. Having two copies of a signing key makes it impossible to identify with certainty who used the key; a second archived copy could be used to impersonate the digital identity of the original key owner.

Clients that generate single key pairs use the same private key for both signing and encrypting data, so a private key derived from a single key pair cannot be archived and recovered. Clients that can generate dual key pairs use one private key for encrypting data and the other for signing data. Since the private encryption key is separate, it can be archived.

In addition to generating dual key pairs, the clients must also support archiving the encryption key in certificate requests. This option archives keys at the time the private encryption keys are generated as a part of issuing the certificate.

### 2.1.4.1. Archiving Keys

The DRM automatically archives private encryption keys if archiving is configured.

If an end entity loses a private encryption key or is unavailable to use the private key, the key must be recovered before any data that was encrypted with the corresponding public key can be read. Recovery is possible if the private key was archived when the key was generated.

There are some common situations when it is necessary to recover encryption keys:

- An employee loses the private encryption key and cannot read encrypted mail messages.
- An employee is on an extended leave, and someone needs to access an encrypted document.
- An employee leaves the company, and company officials need to perform an audit that requires gaining access to the employee's encrypted mail.

The DRM stores private encryption keys in a secure key repository in its internal database; each key is encrypted and stored as a key record and is given a unique key identifier.

The archived copy of the key remains wrapped with the DRM's storage key. It can be decrypted, or unwrapped, only by using the corresponding private key pair of the storage certificate. A combination of one or more key recovery (or DRM) agents' certificates authorizes the DRM to complete the key recovery to retrieve its private storage key and use it to decrypt/recover an archived private key.

The DRM indexes stored keys by key number, owner name, and a hash of the public key, allowing for highly efficient searching. The key recovery agents have the privilege to insert, delete, and search for key records.

- When the key recovery agents search by the key ID, only the key that corresponds to that ID is returned.
- When the agents search by user name, all stored keys belonging to that owner are returned.
- When the agents search by the public key in a certificate, only the corresponding private key is returned.

When a Certificate Manager receives a certificate request that contains the key archival option, it automatically forwards the request to the DRM to archive the encryption key. The private key is encrypted by the transport key, and the DRM receives the encrypted copy and stores the key in its key repository. To archive the key, the DRM uses two special key pairs:

- A transport key pair and corresponding certificate.
- A storage key pair.

Figure 2.1, “How the Key Archival Process Works” illustrates how the key archival process occurs when an end entity requests a certificate.

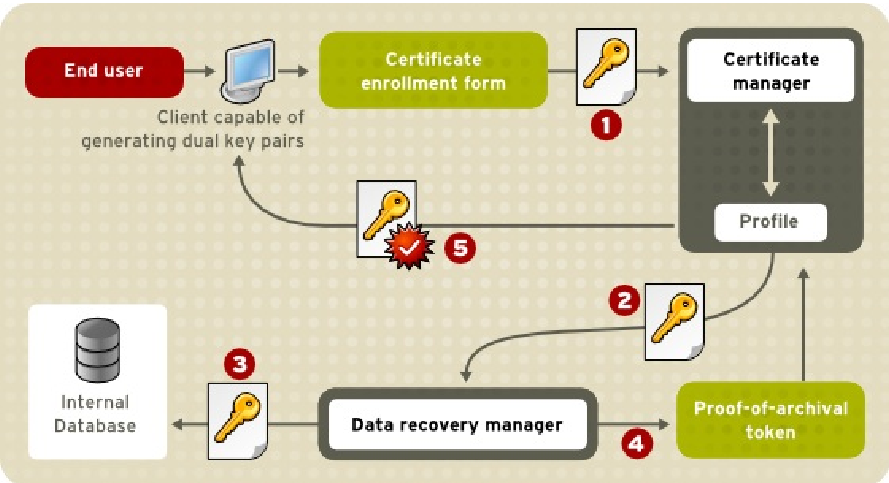



**Figure 2.1. How the Key Archival Process Works**

1. The client requests and generates a dual key pair.
   a. The end entity, using a client which can generate dual key pairs, submits a request through the Certificate Manager enrollment form.
   b. The client detects the JavaScript in the enrollment form and exports only the private encryption key, not the private signing key.
   c. The Certificate Manager detects the key archival option in the request and asks the client for the private encryption key.
   d. The client encrypts the private encryption key with the public key from the DRM's transport certificate embedded in the enrollment form.
2. After approving the certificate request and issuing the certificate, the Certificate Manager sends it to the DRM for storage, along with the public key. The Certificate Manager waits for verification from the DRM that the private key has been received and stored and that it corresponds to the public encryption key.
3. The DRM decrypts it with the private key. After confirming that the private encryption key corresponds to the public encryption key, the DRM encrypts it again with its public key pair of the storage key before storing it in its internal database.
4. Once the private encryption key has been successfully stored, the DRM uses the private key of its transport key pair to sign a token confirming that the key has been successfully stored; the DRM then sends the token to the Certificate Manager.
5. The Certificate Manager issues two certificates for the signing and encryption key pairs and returns them to the end entity.

Both subsystems subject the request to configured certificate profile constraints at appropriate stages. If the request fails to meet any of the profile constraints, the subsystem rejects the request.

#### 2.1.4.2. Key Recovery

The DRM supports agent-initiated key recovery. Agent-initiated recovery is when designated recovery agents use the key recovery form on the DRM agent services page to process and approve key recovery requests. With the approval of a specified number of agents, an organization can recover keys when the key's owner is unavailable or when keys have been lost.

Through the DRM agent services page, key recovery agents can collectively authorize and retrieve private encryption keys and associated certificates in a PKCS #12 package, which can then be imported into the client. (This is explained in more detail in the *Certificate System Agent's Guide*.) To authorize key recovery, the required number of recovery agents access the DRM agent services page and use the **`Authorize Recovery`** button to enter each authorization separately.

In key recovery authorization, one of the key recovery agents informs all required recovery agents about an impending key recovery. All recovery agents access the DRM key recovery page. One of the agents initiates the key recovery process. The DRM returns a notification to the agent includes a recovery authorization reference number identifying the particular key recovery request that the agent is required to authorize. Each agent uses the reference number and authorizes key recovery separately.

The DRM informs the agent who initiated the key recovery process of the status of the authorizations.

**NOTE**

The page that the first agent used to initiate the key recovery request keeps refreshing until all agents required to authorize have performed the authorization. It is important that the first agent does not close this browser session until the authorization is complete. Otherwise, the key recovery request needs to be started again.

When all of the authorizations are entered, the DRM checks the information. If the information presented is correct, it retrieves the requested key and returns it along with the corresponding certificate in the form of a PKCS #12 package to the agent who initiated the key recovery process.

**WARNING**

The PKCS #12 package contains the private key. To minimize the risk of key compromise, the recovery agent must use a secure method to deliver the PKCS #12 package and password to the key recipient. The agent should use a good password to encrypt the PKCS #12 package and set up an appropriate delivery mechanism.

The *key recovery agent scheme* configures the DRM to recognize to which group the key recovery agents belong and specifies how many of these agents are required to authorize a key recovery request before the archived key is restored.

### 2.1.5. About the Token Processing System (TPS)

The Token Processing System (TPS) serves as the conduit between the Enterprise Security Client and the other subsystems (CA, TKS, DRM) in the Certificate System and is the only means for the client to communicate with the other subsystems. It provides the following functionalities for users managing their smart cards through the Enterprise Security Client:

- Working with multiple instances of a subsystem
- Formatting smart cards
- Resetting the PIN on smart card tokens
- Upgrading the applet for smart card tokens
- Enrolling smart cards through the Enterprise Security Client
- Performing LDAP authentication
- Managing the token database
- Logging token events

The TPS must be configured to work with two Certificate System subsystems: the CA, which will process all of the certificate enrollment and revocation requests initiated through the Enterprise Security Client, and the Token Key Service, which generates a master key which is used to derive secret keys specific to each smart card, which are used to wrap (encrypt) the certificates and commands transmitted between the TPS and the client. The TPS can be optionally configured to work with a DRM instance, which will perform server-side key generation and key archival and recovery for the keys and certificates stored on the smart cards.

### 2.1.6. About the Token Key Service (TKS)

The Certificate System Token Management System consists of three components, the Token Processing System (TPS), the Token Key Service (TKS), and the Enterprise Security Client. This section

explains the TKS, which manages the master keys required set up a secure communication channel between the TPS and the client.

A TKS manages the master and transport keys required to generate and distribute keys for smart cards or tokens. A master key is a Triple DES symmetric key stored either in software or hardware token. When supplied with the token CUID, a TKS can derive the corresponding three symmetric keys - authentication key, Mac key, and key encryption key (KEK) - on each token. This effectively shares secrets between the Certificate System and the token without having to store these symmetric keys on the server.

The Certificate System TPS subsystem uses the TKS subsystem to generate the token keys the TPS uses to communicate with the Enterprise Security Client. The TPS communicates with the TKS over SSL. The TKS provides the security between tokens and the TPS since the security relies on the relationship between the master key and the token keys.

The functions provided by the TKS include the following:

- Helps establish a secure channel (signed and encrypted) between the token and TPS.
- Provides proof of presence for the security token during enrollment.
- Supports key changeover when the master key changes on the TKS. Tokens with older keys get new token keys.
- Helps generate a symmetric session key for the DRM to wrap (encrypt) the entity's private key for (optional) server-side key generation, where the entity's encryption keys are generated on the DRM

> **NOTE**
>
> Because of the sensitivity of the data that the TKS manages, the TKS should be set behind the firewall with restricted access.

## 2.2. Red Hat Certificate System Services

There are three different interfaces for managing certificates and subsystems, depending on the user type: administrators, agents, and end users.

### 2.2.1. Interfaces for Administrators

The administrative interface is used to manage the subsystem itself. This includes adding users, configuring logs, managing profiles and plug-ins, and the internal database, among many other functions. This interface is also the only interface that does not directly deal with certificates, tokens, or keys, meaning it is not used for managing the *PKI*, only the *servers*.

There are two types of administrative consoles, Java-based and HTML-based. Although the interface is different, both are accessed using a server URL and the administrative port number.

#### 2.2.1.1. The Java Administrative Console for CA, OCSP, DRM, and TKS Subsystems

The Java console is used by four subsystems: the CA, OCSP, DRM, and TKS. The console is accessed using a locally-installed **pkiconsole** utility. It can access any subsystem because the command requires the hostname, the subsystem's administrative SSL port, and the specific subsystem type.

```
pkiconsole https://server.example.com:admin_port/subsystem_type
```

This opens a console, as in Figure 2.2, “Certificate System Console”.

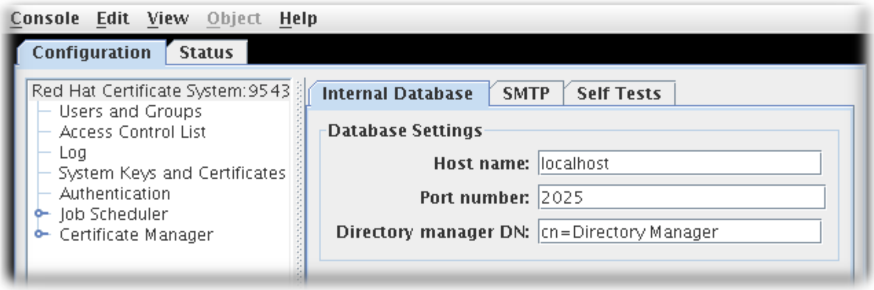



**Figure 2.2. Certificate System Console**

The **`Configuration`** tab controls all of the setup for the subsystem, as the name implies. The choices available in this tab are different depending on which subsystem type the instance is; the CA has the most options since it has additional configuration for jobs, notifications, and certificate enrollment authentication.

All subsystems have four basic options:

- Users and groups
- Access control lists
- Log configuration
- Subsystem certificates (meaning the certificates issued to the subsystem for use, for example, in the security domain or audit signing)

The **`Status`** tab shows the logs maintained by the subsystem.

### 2.2.1.2. The Administrative Interface for the RA and TPS

The RA and TPS subsystems use HTML-based administrative interfaces. These are accessed by entering the hostname and secure port as the URL, authenticating with the administrator's certificate, and clicking the appropriate **`Administrators`** link.

> **NOTE**
>
> There is a single SSL port for RA and TPS subsystems which is used for both administrator and agent services. Access to those services is restricted by certificate-based authentication. The other subsystems used separate SSL ports for the agent and administrative services, along with certificate-based authentication.

The HTML admin interface is much more limited than the Java console; the primary administrative function is managing the subsystem users; all other administrative tasks are done by manually editing the **`CS.cfg`** file.

The RA allows administrators to create and edit users and groups for the subsystem.


**Figure 2.3. RA Admin Page**

The TPS only allows operations to manage users for the TPS subsystem. However, the TPS admin page can also list tokens and display all activities (including normally-hidden administrative actions) performed on the TPS.


**Figure 2.4. TPS Admin Page**

## 2.2.2. Agent Interfaces

The agent services pages are where almost all of the certificate and token management tasks are performed. These services are HTML-based, and agents authenticate to the site using a special agent certificate.

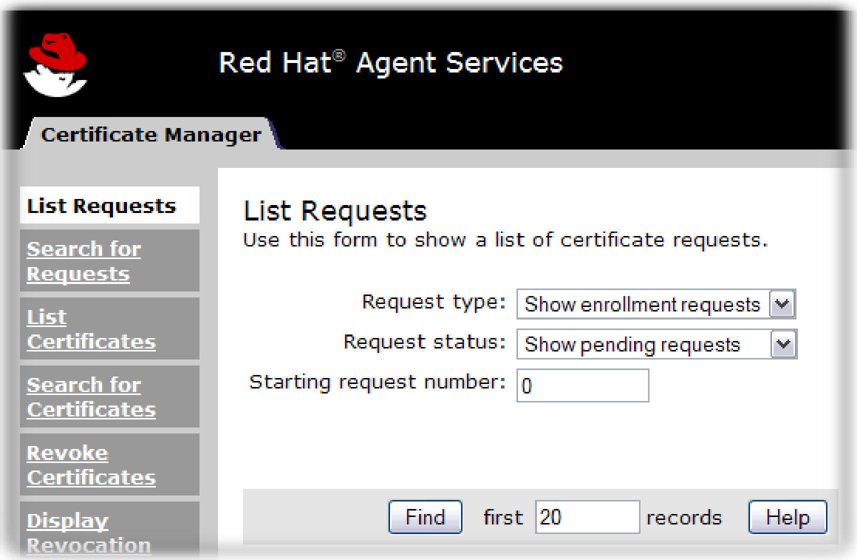



**Figure 2.5. Certificate Manager's Agent Services Page**

The operations vary depending on the subsystem:

- The Certificate Manager agent services include approving certificate requests (which issues the certificates), revoking certificates, and publishing certificates and CRLs. All certificates issued by the CA can be managed through its agent services page.
- The TPS agent services, like the CA agent services, manages all of the tokens which have been formatted and have had certificates issued to them through the TPS. Tokens can be enrolled, suspended, and deleted by agents. Two other roles (operator and admin) can view tokens in web services pages, but cannot perform any actions on the tokens.
- DRM agent services pages process key recovery requests, which set whether to allow a certificate to be issued reusing an existing key pair if the certificate is lost.
- The OCSP agent services page allows agents to configure CAs which publish CRLs to the OCSP, to load CRLs to the OCSP manually, and to view the state of client OCSP requests.
- The RA agent services allows agents to list and approve certificate requests and to check the status of requests and certificates processed through the RA.

The TKS is the only subsystem without an agent services page.

### 2.2.3. End User Pages

The CA, RA, and TPS all process direct user requests in some way. That means that end users have to have a way to connect with those subsystems. The CA and RA both have end-user, or *end-entities*, HTML services. The TPS uses the Enterprise Security Client.

The end-user services are accessed over standard HTTP using the server's hostname and the standard port number; they can also be accessed over HTTPS using the server's hostname and the specific end-entities SSL port.

For CAs, each type of SSL certificate is processed through a specific online submission form, called a *profile*. There are about two dozen certificate profiles for the CA, covering all sorts of certificates — user

SSL certificates, server SSL certificates, log and file signing certificates, email certificates, and every kind of subsystem certificate. There can also be custom profiles.


**Figure 2.6. Certificate Manager's End-Entities Page**

End users retrieve their certificates through the CA pages when the certificates are issued. They can also download CA chains and CRLs and can revoke or renew their certificates through those pages.

The RA is a more lightweight subsystem, so it only processes four common certificate profiles. Like the CA, the enrollment forms are accessed through the `End Entities` URL. Users can submit certificate requests and retrieve their certificates through the RA.

## 2.2.4. Enterprise Security Client

The **Enterprise Security Client** is a tool for Red Hat Certificate System which simplifies managing smart cards. End users can use security tokens (smart cards) to store user certificates used for applications such as single sign-on access and client authentication. End users are issued the tokens containing certificates and keys required for signing, encryption, and other cryptographic functions.

The **Enterprise Security Client** is the third part of Certificate System's complete token management system. Two subsystems — the Token Key Service (TKS) and Token Processing System (TPS) — are used to process token-related operations. The **Enterprise Security Client** is the interface which allows the smart card and user to access the token management system.

After a token is enrolled, applications such as Mozilla Firefox and Thunderbird can be configured to recognize the token and use it for security operations, like client authentication and S/MIME mail. Enterprise Security Client provides the following capabilities:

- Supports JavaCard 2.1 or higher cards and Global Platform 2.01-compliant smart cards like Safenet's 330J smart card
- Supports Global Platform 2.01-compliant smart cards like Gemalto e-gate 32K and Gemalto TOP IM FIPS CY2 tokens, both the smart card and GemPCKey USB form factor key.
- Enrolls security tokens so they are recognized by TPS.
- Maintains the security token, such as re-enrolling a token with TPS.
- Provides information about the current status of the token or tokens being managed.
- Supports server-side key generation so that keys can be archived and recovered on a separate token if a token is lost.

The Enterprise Security Client is a cross-platform client for end users to register and manage keys and certificates on smart cards or tokens. This is the final component in the Certificate System token management system, with the Token Processing System (TPS) and Token Key Service (TKS).

**NOTE**

For more information on using the Enterprise Security Client, see the *Certificate System Enterprise Security Client Guide*.

The Enterprise Security Client provides the user interface of the token management system. The end user can be issued security tokens containing certificates and keys required for signing, encryption, and other cryptographic functions. To use the tokens, the TPS must be able to recognize and communicate with them. Enterprise Security Client is the method for the tokens to be enrolled.

Enterprise Security Client communicates over an SSL HTTP channel to the backend of the TPS. It is based on an extensible Mozilla XULRunner framework for the user interface, while retaining a legacy web browser container for a simple HTML-based UI.

After a token is properly enrolled, web browsers can be configured to recognize the token and use it for security operations. Enterprise Security Client provides the following capabilities:

- Allows the user to enroll security tokens so they are recognized by the TPS.
- Allows the user to maintain the security token. For example, Enterprise Security Client makes it possible to re-enroll a token with the TPS.
- Provides support for several different kinds of tokens through default and custom token profiles. By default, the TPS can automatically enroll user keys, device keys, and security officer keys; additional profiles can be added so that tokens for different uses (recognized by attributes such as the token CUID) can automatically be enrolled according to the appropriate profile.
- Provides information about the current status of the tokens being managed.

# Chapter 3. Supported Standards and Protocols

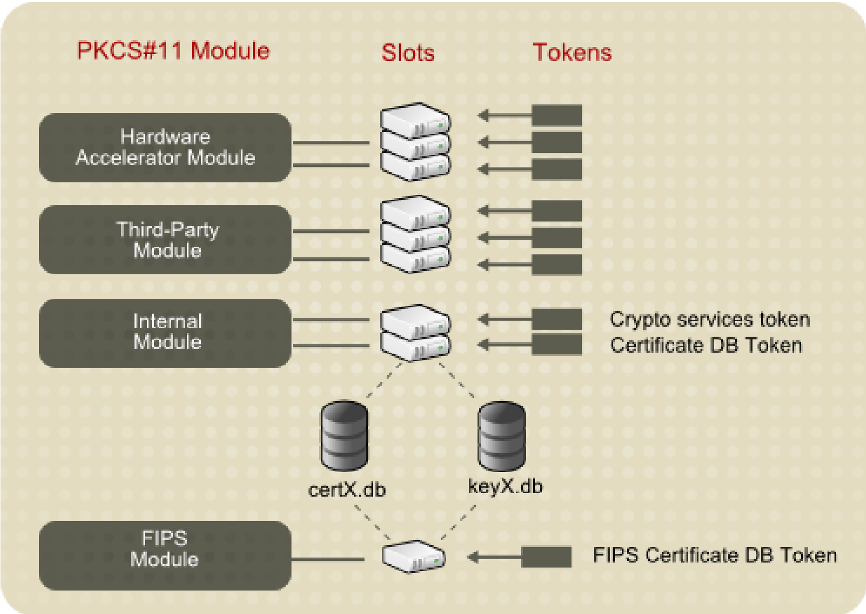
Red Hat Certificate System is based on many public and standard protocols and RFCs, to ensure the best possible performance and interoperability. The major standards and protocols used or supported by Certificate System 8.0 are outlined in this chapter, to help administrators plan their client services effectively.

## 3.1. PKCS #11

Public-Key Cryptography Standard (PKCS) #11 specifies an API used to communicate with devices that hold cryptographic information and perform cryptographic operations. Because it supports PKCS #11, Certificate System is compatible with a wide range of hardware and software devices.

At least one PKCS #11 module must be available to any Certificate System subsystem instance. A PKCS #11 module (also called a cryptographic module or cryptographic service provider) manages cryptographic services such as encryption and decryption. PKCS #11 modules are analogous to drivers for cryptographic devices that can be implemented in either hardware or software. Certificate System contains a built-in PKCS #11 module and can support third-party modules.

A PKCS #11 module always has one or more slots which can be implemented as physical hardware slots in a physical reader such as smart cards or as conceptual slots in software. Each slot for a PKCS #11 module can in turn contain a token, which is the hardware or software device that actually provides cryptographic services and optionally stores certificates and keys.




Two cryptographic modules are included in the Certificate System:

- The default internal PKCS #11 module, which comes with two tokens:
  - The internal crypto services token, which performs all cryptographic operations such as encryption, decryption, and hashing.
  - The internal key storage token ("Certificate DB token"), which handles all communication with the certificate and key database files that store certificates and keys.
- The FIPS 140-1 module. This module complies with the FIPS 140-1 government standard for cryptographic module implementations. The FIPS 140-1 module includes a single, built-in FIPS 140-1 certificate database token, which handles both cryptographic operations and communication with the certificate and key database files.

Any PKCS #11 module can be used with the Certificate System. To use an external hardware token with a subsystem, load its PKCS #11 module before the subsystem is configured, and the new token is available to the subsystem.

Available PKCS #11 modules are tracked in the **`secmod.db`** database for the subsystem. This file can be modified using the **`modutil`** tool, and it should be modified when there are changes to the system like installing hardware accelerators to use for signing operations. For more information on **`modutil`**, see http://www.mozilla.org/projects/security/pki/nss/tools/.

PKCS #11 hardware devices also provide key backup and recovery features for the information stored on the hardware token. Refer to the PKCS #11 vendor documentation for information on retrieving keys from the tokens.

## 3.2. SSL/TLS, ECC, and RSA

Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols are universally accepted standards for authenticated and encrypted communication between clients and servers. Both client and server authentication occur over SSL/TLS.

SSL/TLS uses a combination of public key and symmetric-key encryption. Symmetric-key encryption is much faster than public-key encryption, but public-key encryption provides better authentication techniques. An SSL/TLS session always begins with an exchange of messages called the *SSL handshake*, initial communication between the server and client. The handshake allows the server to authenticate itself to the client using public-key techniques, then allows the client and the server to cooperate in the creation of symmetric keys used for rapid encryption, decryption, and tamper detection during the session that follows.

Both of these protocols support using a variety of different cryptographic algorithms, or *ciphers*, for operations such as authenticating the server and client, transmitting certificates, and establishing session keys. Clients and servers may support different cipher suites, or sets of ciphers. Among other functions, the SSL handshake determines how the server and client negotiate which cipher suite they will use to authenticate each other, to transmit certificates, and to establish session keys.

Key-exchange algorithms like RSA and Elliptic Curve Cryptography (ECC) govern the way the server and client determine the symmetric keys to use during an SSL session. The most common SSL cipher suites use RSA key exchange, while TLS supports ECC cipher suites as well as RSA. The Certificate System [1] supports both RSA and ECC public-key cryptographic systems.

> **NOTE**
>
> Longer RSA keys are required to provide security as computing capabilities increase. The recommended RSA key-length is 2048 bits. Though many web servers continue to use 1024-bit keys, web servers should migrate to at least 2048 bits. For 64-bit machines, consider using stronger keys. *All* CAs should use at least 2048-bit keys, and stronger keys (such as 3072 or 4096 bits) if possible.

As PKIs using RSA keys and certificates transition to other cryptographic systems like ECC, servers should continue to support RSA. Certificate System supports using both RSA- and ECC-based certificates in the same subsystem.

### 3.2.1. Supported Cipher Suites for RSA

Certificate System supports several different cipher suites with the RSA key exchange:

- *AES and SHA-1 Message Authentication*. Advanced Encryption Standard (AES) ciphers have a fixed block size of 128-bits, and the keys can be either 128-bit or 256-bit. There are 3.4 x $10^{38}$ possible 128-bit keys and 1.1 x $10^{77}$ possible 256-bit keys. There are more possible keys than any other cipher, making AES the strongest cipher supported by SSL. These cipher suites are FIPS-compliant.
- *Triple DES and SHA-1 Message Authentication*. Triple DES (Data Encryption Standard) is the second-strongest cipher supported by SSL, but it is not as fast as RC4. Triple DES uses a key three times as long as the key for standard DES. Because the key size is so large, there are approximately 3.7 * $10^{50}$ possible keys. This cipher suite is FIPS-compliant.
- *RC4 and RC2 and MD5 Message Authentication*. The RC4 and RC2 ciphers have 128-bit encryption, which permits approximately 3.4 * $10^{38}$ possible keys. This makes RC4 or RC2 keys very difficult to crack. RC4 ciphers are faster than RC2 ciphers.

  RC4 can use SHA-1 message authentication as well as MD5 message authentication.
- *DES and SHA-1 Message Authentication*. DES 56-bit encryption permits approximately 7.2 * $10^{16}$ possible keys. This cipher suite is no longer FIPS-compliant because it is too weak cryptographically.

### 3.2.2. Using ECC

*Elliptic Curve Cryptography (ECC)* is a cryptographic system that uses elliptic curves to create keys for encrypting data. ECC creates cryptographically-stronger keys with shorter key lengths than RSA, which makes it faster and more efficient to implement.

ECC has several advantages over RSA, since it is faster and requires shorter key lengths for stronger keys. The drawback to using ECC is that it is not as widely supported as RSA.

**Table 3.1. Comparison of RSA and ECC Cipher Strength**

| Bits of Security [a] | RSA Key Length | ECC Key Length |
|---|---|---|
| 80 | 1024 | 160-223 |
| 112 | 2048 | 224-255 |
| 128 | 3072 | 256-383 |
| 192 | 7860 | 384-511 |
| 256 | 15360 | 512+ |

[a] The information in this table is from the National Institute of Standards and Technology (NIST). For more information, see http://csrc.nist.gov/publications/nistpubs/800-57/SP800-57-Part1.pdf.

Certificate System supports using ECC with all of its subsystems, so ECC certificate requests can be submitted to CAs through any of the enrollment profiles and ECC keys can be archived and restored in the DRM. However, Certificate System does not include ECC support natively, so using ECC is slightly different than using RSA:

- An external PKCS#11 module must be loaded before any subsystems are installed so that they can be configured with ECC subsystem certificates.
- The CA profile pages can *process* ECC certificate requests, but they cannot *generate* ECC keys. Some of the profile forms, like manual user certificates, then cannot be used for ECC certificates. In those cases, a different or custom profile needs to be used, and certificate requests have to be generated using **`certutil`**.

For a CA to issue ECC certificates, the CA must be configured with an ECC CA signing certificate. This is best done by loading an ECC PKCS#11 module before the CA is installed, and then configuring the CA using ECC keys.

> **NOTE**
>
> A CA with an ECC CA signing certificate can issue both ECC and RSA certificates. A CA with an RSA CA signing certificate can only issue RSA certificates.

Only the CA signing certificate is required; if for support purposes it is better to use RSA client certificates with the CA, simply delete the ECC subsystem certificates (except for the signing certificate) and replace them with RSA certificates.

For more information on ECC, see RFC 4492, Section 5.6.1, Table 2.

## 3.3. IPv4 and IPv6 Addresses

Certificate System supports both IPv4 addresses and IPv6 addresses. In a very wide variety of circumstances, Certificate System subsystems or operations references a hostname or IP address; supporting both IPv4- and IPv6-style addresses ensures forward compatibility with network protocols. The operations that support IPv6 connections include the following:

- Communications between subsystems, including between the RA and CA and between the TPS, TKS, and CA and for joining security domains
- Token operations between the TPS and Enterprise Security Client
- Subsystem logging
- Access control instructions
- Operations performed with Certificate System tools, including the Subject Alt Name Extension tool, HttpClient, and the Bulk Issuance Tool
- Client communications, including both the **`pkiconsole`** and IPv6-enabled browsers for web services
- Certificate request names and certificate subject names, including user, server, and router certificates
- Publishing
- Connecting to LDAP databases for internal databases and authentication directories

Any time a hostname or URL is referenced, an IP address can be used:

- An IPv4 address must be in the format *n.n.n.n* or *n.n.n.n,m.m.m.m*. For example, *128.21.39.40* or *128.21.39.40,255.255.255.00*.
- An IPv6 address uses a 128-bit namespace, with the IPv6 address separated by colons and the netmask separated by periods. For example, *0:0:0:0:0:0:13.1.68.3*, *FF01::43*, *0:0:0:0:0:0:13.1.68.3,FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:255.255.255.0*, and *FF01::43,FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FF00:0000*.

If DNS is properly configured, then an IPv4 or IPv6 address can be used to connect to the web services pages and to the subsystem Java consoles. The most common method is to use fully-qualified domain names:

```
https://ipv6host.example.com:9445/ca/services
pkiconsole https://ipv6host.example.com:9445/ca
```

To use IPv6 numeric addresses, then replace the fully-qualified domain name in the URL with the IPv6 address, enclosed in brackets (**`[]`**). For example:

```
https://[00:00:00:00:123:456:789:00:]:9445/ca/services

pkiconsole https://[00:00:00:00:123:456:789:00:]:9445/ca
```

## 3.4. Supported PKIX Formats and Protocols

The Certificate System supports many of the protocols and formats defined in Public-Key Infrastructure (X.509) by the IETF. Along with the PKIX standards listed here, other PKIX-listed standards are available at http://www.ietf.org/html.charters/pkix-charter.html under the **`Internet Drafts`** section.

**Table 3.2. PKIX Standards Supported in Certificate System 8.0**

| Format or Protocol | RFC or Draft | Description |
|---|---|---|
| X.509 version 1 and version 3 | | Digital certificate formats recommended by the International Telecommunications Union (ITU). |
| Certificate Request Message Format (CRMF) | RFC 4211 | A message format to send a certificate request to a CA. |
| Certificate Management Message Formats (CMMF) | | Message formats to send certificate requests and revocation requests from end entities to a CA and to return information to end entities. CMMF has been subsumed by another standard, CMC. |
| Certificate Management Messages over CS (CMC) | RFC 5274 | A general interface to public-key certification products based on CS and PKCS #10, including a certificate enrollment protocol for RSA-signed certificates with Diffie-Hellman public-keys. CMC incorporates CRMF and CMMF. |
| Cryptographic Message Syntax (CMS) | RFC 2630 | A superset of PKCS #7 syntax used for digital signatures and encryption. |
| PKIX Certificate and CRL Profile | RFC 5280 | A standard developed by the IETF for a public-key infrastructure for the Internet. It specifies profiles for certificates and CRLs. For more information about certificate and CRL profiles, see http://www.ietf.org/rfc/rfc5280.txt. |

## 3.5. Supported Security and Directory Protocols

The Certificate System supports several common Internet and network protocols.

**Table 3.3. Supported Security and Directory Protocols**

| Protocol | Description |
|---|---|
| FIPS PUBS 140-1 | Federal Information Standards Publications (FIPS PUBS) 140-1 is a US government standard for implementing cryptographic modules such as hardware or software that encrypts and decrypts data, creates and verifies digital signatures, and other cryptographic functions. |
| Hypertext Transport Protocol (HTTP) and Hypertext Transport Protocol Secure (HTTPS) | Protocols used to communicate with web servers. |
| KEYGEN tag | An HTML tag that generates a key pair for use with a certificate. |
| Lightweight Directory Access Protocol (LDAP) v2, v3 | A directory service protocol designed to run over TCP/IP and across multiple platforms. LDAP is a simplified version of Directory Access Protocol (DAP), used to access X.500 directories. LDAP is under IETF change control and has evolved to meet Internet requirements. |
| Public-Key Cryptography Standard (PKCS) #7 | An encrypted data and message format developed by RSA Data Security to represent digital signatures, certificate chains, and encrypted data. This format is used to deliver certificates to end entities. |
| Public-Key Cryptography Standard (PKCS) #10 | A message format developed by RSA Data Security for certificate requests. This format is supported by many server products. |
| Public-Key Cryptography Standard (PKCS) #11 | Specifies an API used to communicate with devices such as hardware tokens that hold cryptographic information and perform cryptographic operations. |
| Secure Sockets Layer (SSL) 2.0 and 3.0 and Transport Layer Security (TLS) | A set of rules governing server authentication, client authentication, and encrypted communication between servers and clients. |
| Security-Enhanced Linux | Security-enhanced Linux, or SELinux, is a set of security protocols enforcing mandatory access control on Linux system kernels. This was developed by the United States National Security Agency to keep applications from accessing confidential or protected files through lenient or flawed access controls. |
| Simple Certificate Enrollment Protocol (SCEP) | A protocol designed by Cisco to specify a way for a router to communicate with an RA or CA for router certificate enrollment. SCEP defines two modes of operation: RA mode and CA mode. Certificate System supports CA mode, where the the request is encrypted with the CA signing certificate. |
| UTF-8 | The certificate enrollment pages support all UTF-8 characters for specific fields (common name, organizational unit, requester name, and |

| | |
|---|---|
| | additional notes). The certificates will be generated with the UTF-8 strings correctly used in the subject names and other fields, and the UTF-8 strings are searchable and correctly display in the CA, OCSP, and DRM end user and agents services pages.<br>This UTF-8 support does not extended to internationalized domain names, like in email addresses. |
| IPv4 and IPv6 | Certificate System supports both IPv4 and IPv6 address namespaces for communications and operations with all subsystems and tools, as well as for clients, subsystem creation, and token and certificate enrollment, |

[1] Only RSA is supported natively. External PKCS#11 modules with ECC support must be loaded for subsystems to use ECC.

# Chapter 4. Major Features in Certificate System

This chapter covers some of the major features of Red Hat Certificate System, giving a brief rundown of the major functionality of the product. These summaries are meant to help administrators understand the capabilities of the Certificate System so they can effectively plan their subsystem deployments.

## 4.1. Certificate Issuance

The Certificate System supports enrolling and issuing certificates and processing certificate requests from a variety of end entities, such as web browsers, servers, and virtual private network (VPN) clients. Issued certificates conform to X.509 version 3 standards.

The Certificate Manager can issue certificates with the following characteristics:

- Certificates that are X.509 version 3-compliant
- Unicode support for the certificate subject name and issuer name
- Support for empty certificate subject names
- Support for customized subject name components
- Support for customized extensions

Additionally, smart cards can have certificates enrolled and maintained through the Enterprise Security Client. The Enterprise Security Client communicates directly with the TPS system, which, in turn, processes requests through the CA and DRM subsystems. Certificates are generated automatically when the token is first formatted, and all additional certificates belonging to the user can be imported onto the token.

## 4.2. Authentication for Certificate Enrollment

Certificate System provides authentication options for certificate enrollment. These include agent-approved enrollment, in which an agent processes the request, and automated enrollment, in which an authentication method is used to authenticate the end entity and then the CA automatically issues a certificate. CMC enrollment is also supported, which automatically processes a request approved by an agent.

## 4.3. Certificate Profiles

The Certificate System uses certificate profiles to configure the content of the certificate, the constraints for issuing the certificate, the enrollment method used, and the input and output forms for that enrollment. A single certificate profile is associated with issuing a particular type of certificate.

A set of certificate profiles is included for the most common certificate types; the profile settings can be modified. Certificate profiles are configured by an administrator, and then sent to the agent services page for agent approval. Once a certificate profile is approved, it is enabled for use. A dynamically-generated HTML form for the certificate profile is used in the end-entities page for certificate enrollment, which calls on the certificate profile. The server verifies that the defaults and constraints set in the certificate profile are met before acting on the request and uses the certificate profile to determine the content of the issued certificate.

## 4.4. CRLs

The Certificate System can create certificate revocation lists (CRLs) from a configurable framework which allows user-defined issuing points so a CRL can be created for each issuing point. Delta CRLs

can also be created for any issuing point that is defined. CRLs can be issued for each type of certificate, for a specific subset of a type of certificate, or for certificates generated according to a profile or list of profiles. The extensions used and the frequency and intervals when CRLs are published can all be configured.

The Certificate Manager issues X.509-standard CRLs. A CRL can be automatically updated whenever a certificate is revoked or at specified intervals.

## 4.5. Publishing

Certificates can be published to files and an LDAP directory, and CRLs to files, an LDAP directory, and an OCSP responder. The publishing framework provides a robust set of tools to publish to all three places and to set rules to define with more detail which types of certificates or CRLs are published where.

## 4.6. Notifications

The notification feature sets up automated messages when a particular event occurs, such as when a certificate is issued or revoked. The notification framework comes with default modules that can be enabled and configured.

## 4.7. Jobs

The jobs feature sets up automated jobs that run at defined intervals.

## 4.8. Dual Key Pairs

The Certificate System supports generating dual key pairs, separate key pairs for signing and encrypting email messages and other data. To support separate key pairs for signing and encrypting data, dual certificates are generated for end entities, and the encryption keys are archived. If a client makes a certificate request for dual key pairs, the server issues two separate certificates.

## 4.9. Cross-Pair Certificates

It is possible to create a trusted relationship between two separate CAs by issuing and storing cross-signed certificates between these two CAs. By using cross-signed certificate pairs, certificates issued outside the organization's PKI can be trusted within the system.

## 4.10. Logging

The Certificate System and each subsystem produce extensive system and error logs that record system events so that the systems can be monitored and debugged. All log records are stored in the local file system for quick retrieval. Logs are configurable, so logs can be created for specific types of events and at the desired logging level.

Certificate System allows logs to be signed digitally before archiving them or distributing them for auditing. This feature enables log files to be checked for tampering after being signed.

## 4.11. Auditing

The Certificate System maintains audit logs for all events, such as requesting, issuing and revoking

certificates and publishing CRLs. These logs are then signed. This allows authorized access or activity to be detected. An outside auditor can then audit the system if required. The assigned auditor user account is the only account which can view the signed audit logs. This user's certificate is used to sign and encrypt the logs. Audit logging is configured to specify the events that are logged.

## 4.12. Self-Tests

The Certificate System provides the framework for system self-tests that are automatically run at startup and can be run on demand. A set of configurable self-tests are already included with the Certificate System.

## 4.13. Access Controls

Certificate System users can be assigned to groups, and they then have the privileges of whichever group they are members. A user only has privileges for the instance of the subsystem in which the user is created and the privileges of the group to which the user is a member.

The Certificate System provides an authorization framework for creating groups and assigning access control to those groups. The default access control on preexisting groups can be modified, and access control can be assigned to individual users and IP addresses. Access points for authorization have been created for the major portions of the system, and access control rules can be set for each point.

The Certificate System is configured by default with four user types with different access levels to the system:

- *Administrators*, who can perform any administrative or configuration task for a subsystem.
- *Agents*, who perform PKI management tasks, like approving certificate requests, managing token enrollments, or recovering keys.
- *Auditors*, who can view and configure audit logs.
- *Trusted managers*, which are subsystems with trusted relationship with another subsystem.

Additionally, when a security domain is created, the CA subsystem which hosts the domain is automatically granted the role of *Security Domain Administrator*, which gives the subsystem the ability to manage the security domain and the subsystem instances within it. Other security domain administrator roles can be created for the different subsystem instances.

## 4.14. Security-Enhanced Linux Support

SELinux is a collection of mandatory access control rules which are enforced across a system to restrict unauthorized access and tampering. SELinux is described in more detail in the Red Hat Enterprise Linux documentation, such as "Introduction to SELinux" in the Red Hat Enterprise Linux *Deployment Guide*.

Basically, SELinux identifies *objects* on a system, which can be files, directories, users, processes, sockets, or any other resource on a Linux host. These objects correspond to the Linux API objects. Each object is then mapped to a *security context*, which defines the type of object and how it is allowed to function on the Linux server.

Objects can be grouped into domains, and then each domain is assigned the proper rules. Each security context has rules which set restrictions on what operations it can perform, what resources it can access, and what permissions it has.

The Certificate System has a separate RPM of SELinux policies installed by default. These SELinux policies apply to every subsystem and service used by Certificate System. By running Certificate System

with SELinux in enforcing mode, the security of the information created and maintained by Certificate System is enhanced.

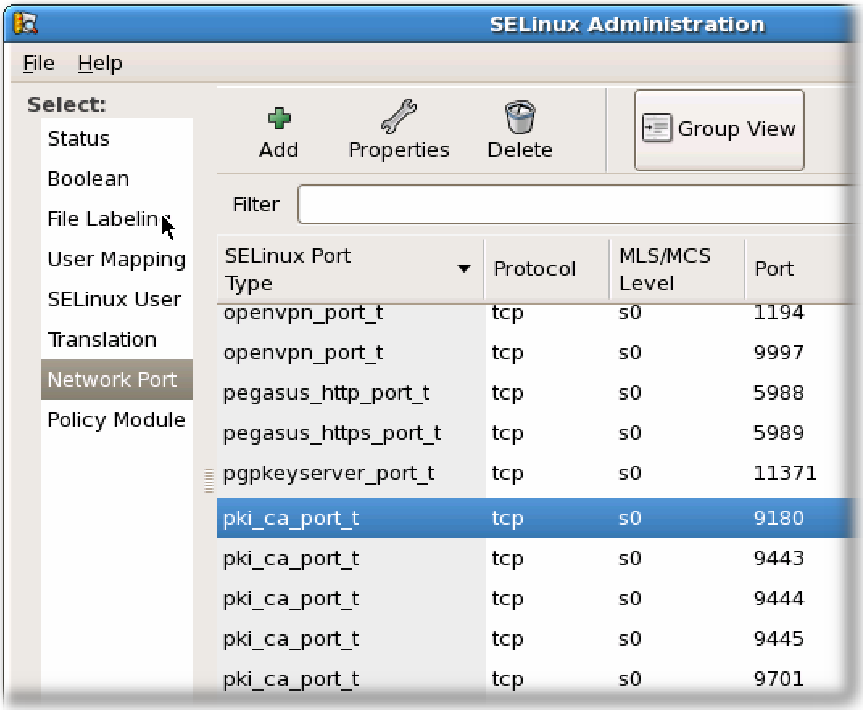



**Figure 4.1. CA SELinux Port Policy**

The Certificate System SELinux policies define the SELinux configuration for every subsystem instance:

- Files and directories for each subsystem instance are labeled with a specific SELinux context.
- The ports for each subsystem instance are labeled with a specific SELinux context.
- All Certificate System processes are constrained within a subsystem-specific domain.
- Each domain has specific rules that define what actions that are authorized for the domain.
- Any access not specified in the SELinux policy is denied to the Certificate System instance.

For Certificate System, each subsystem is treated as an SELinux object, and each subsystem has unique rules assigned to it. The defined SELinux policies allow Certificate System objects run with SELinux set in enforcing mode.

Every time **`pkicreate`** is run, new SELinux policies are automatically configured for the instance. All SELinux policies are updated every time a subsystem is added with **`pkicreate`** or removed with **`pkiremove`**.

The central definition for each instance is its SELinux domain. Each Certificate System subsystem runs in a single subsystem-specific SELinux domain, no matter how many subsystems are installed on a host. For example, if there are three CAs installed on a server, all three belong to the **`pki_ca_t`** SELinux domain.

Each SELinux policy sets rules on what actions the instance is allowed to perform on the system, based on the domain to which the instance belongs. For example, instances in the CA domain (**`pki_ca_t`**) are limited to write access for files with the CA context (**`pki_ca_var_log_t`**) and to access ports that match the CA type (**`pki_ca_port`**). When each Certificate System process is started, it initially runs in an unconfined domain (**`unconfined_t`**) and then transitions into the appropriate subsystem-specific

domain.

The SELinux mode can be changed from enforcing to permissive, or even off, though this is not recommended.

# Chapter 5. Planning the Certificate System

Each Red Hat Certificate System subsystem is installed and configured separately. They can all be installed on the same machine, installed on separate servers, or have multiple instances installed across an organization. Before installing any subsystem, it is important to plan the deployment out: what kind of PKI services do you need? What are the network requirements? What people need to access the Certificate System, what are their roles, and what are their physical locations? What kinds of certificates do you want to issue and what constraints or rules need to be set for them?

This chapter covers some basic questions for planning a Certificate System deployment. Many of these decisions are interrelated; one choice impacts another, like deciding whether to use smart cards determines whether to install the TPS and TKS subsystems.

## 5.1. Deciding on the Required Subsystems

The Certificate System subsystems cover different aspects of managing certificates. Planning which subsystems to install is one way of defining what PKI operations the deployment needs to perform.

Certificates, like software or equipment, have a lifecycle with defined stages. At its most basic, there are three steps:

- It is requested and issued.
- It is valid.
- It expires.

However, this simplified scenario does not cover a lot of common issues with certificates:

- What if an employee leaves the company before the certificate expires?
- When a CA signing certificate expires, all of the certificates issued and signed using that certificate also expire. So will the CA signing certificate be renewed, allowing its issued certificates to remain valid, or will it be reissued?
- What if an employee loses a smart card or leaves it at home. Will a replacement certificate be issued using the original certificates keys? Will the other certificates be suspended or revoked? Are temporary certificates allowed?
- When a certificate expires, will a new certificate be issued or will the original certificate be renewed?

This introduces three other considerations for managing certificates: revocation, renewal, and replacements.

Other considerations are the loads on the certificate authority. Are there a lot of issuance or renewal requests? Is there a lot of traffic from clients trying to validate whether certificates are valid? How are people requesting certificates supposed to authenticate their identity, and does that process slow down the issuance process?

### 5.1.1. Single Certificate Manager

The core of the Certificate System PKI is the Certificate Manager, a certificate authority. The CA receives certificate requests and issues all certificates.

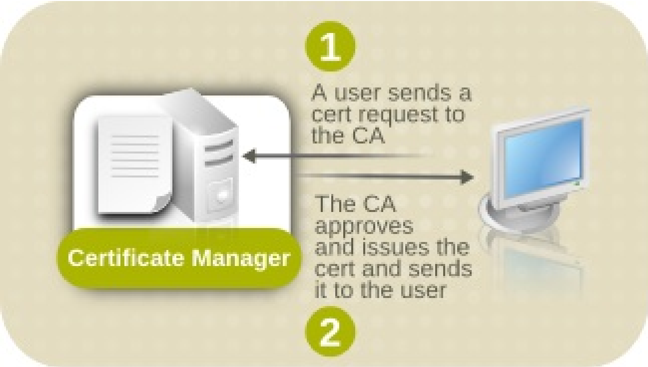
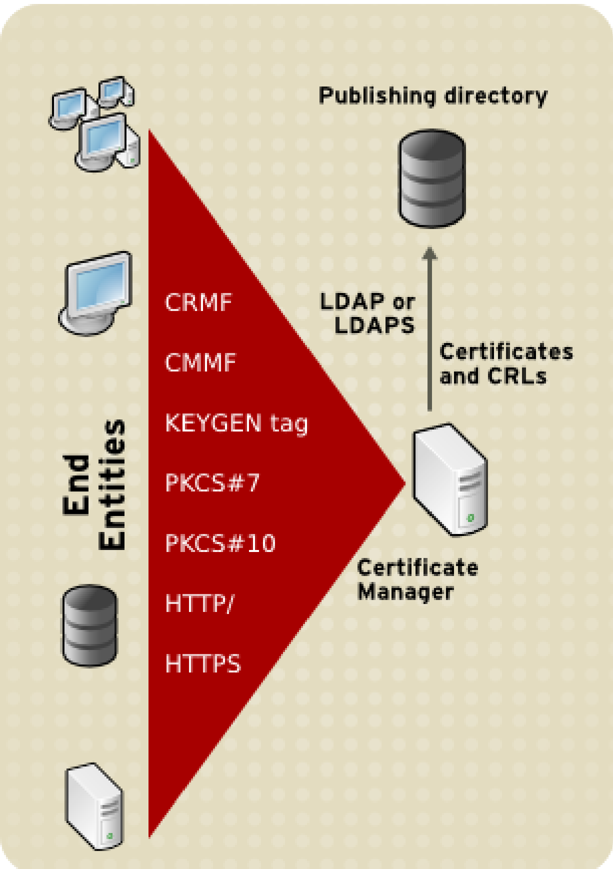



**Figure 5.1. CA Only Certificate System**

All of the basic processing for requests and issuing certificates can handled by the Certificate Manager, and it is the only required subsystem. There can be a single Certificate Manager or many Certificate managers, arranged in many different relationships, depending on the demands of the organization.

Along with issuing certificates, a Certificate Manager can also revoke certificates. One question for administrators is how to handle certificates if they are lost, compromised, or if the person or equipment for which they are issued leaves the company. Revoking a certificate invalidates it before its expiration date, and a list of revoked certificates is compiled and published by the issuing CA so that clients can verify the certificate status.




## 5.1.2. Planning for Lost Keys: Key Archival and Recovery

One operation the CA cannot perform, though, is key archival and recovery. A very real scenario is that a user is going to lose his private key — for instance, the keys could be deleted from a browser database or a smart card could be left at home. Many common business operations use encrypted data, like

encrypted email, and losing the keys which unlock that data means the data itself is lost. Depending on the policies in the company, there probably has to be a way to recover the keys in order to regenerate or reimport a replacement certificate, and both operations require the private key.

That requires a DRM, the subsystem which specially archives and retrieves keys.

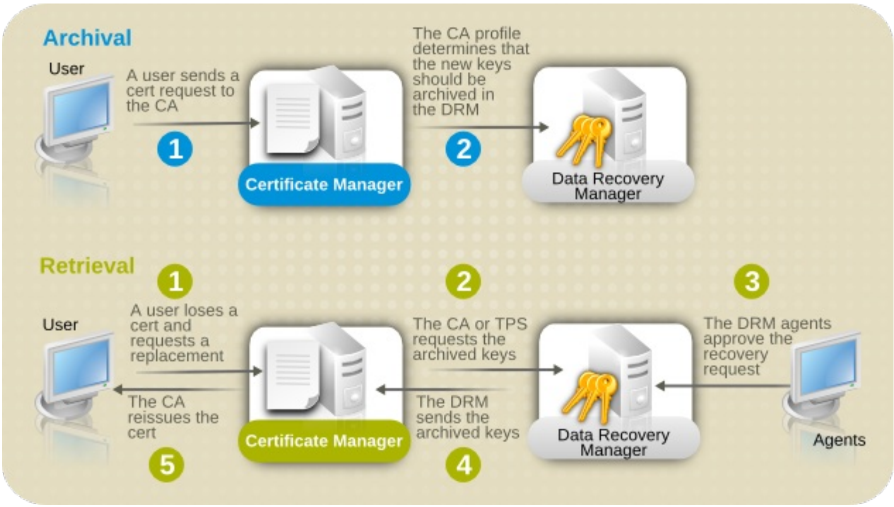



**Figure 5.2. CA and DRM**

The Data Recovery Manager stores encryption keys (key archival) and can retrieve those keys so that the CA can reissue a certificate (key recovery). A DRM can store keys for any certificate generated by Certificate System, whether it is done for a regular certificate or when enrolling a smart card.

The key archival and recovery process is explained in more detail in Section 2.1.4, "About the Data Recovery Manager (DRM)".

**NOTE**

The DRM is intended for archival and recovery of private encryption keys only. Therefore, end users must use a browser that supports dual-key generation to store their public-private key pairs.

### 5.1.3. Balancing Certificate Request Processing

Another aspect of how the subsystems work together is load balancing. If a site has high traffic, then it is possible to simply install a lot of CAs, as clones of each other or in a flat hierarchy (where each CA is independent) or in a tree hierarchy (where some CAs are subordinate to other CAs); this is covered more in Section 5.2, "Defining the Certificate Authority Hierarchy".

Another option, though is to distribute some of the tasks of a single CA to another subsystem. For example, Example Corp. has a manageable number of people requesting certificates for a single CA to issue. However, because of their security policies, each certificate request has to be verified in person by an agent, with supporting documentation. This creates a bottleneck for the CA agents to approve requests. A registration authority (RA) is installed at each local office; the requests are processed and approved locally, and then a central CA issues all of the certificates.

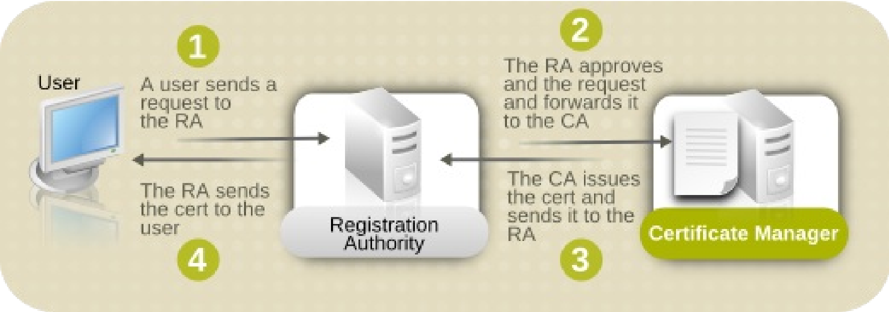



**Figure 5.3. CA and RA**

The Registration Manager takes the load of processing certificate requests; the CA then only has to issue the requests. For network environments where there are strict, and possibly time-consuming, rules for issuing certificates, an RA can speed the process while also give control to local managers and administrators.

RA managers are also good for certain network demands. CAs require a very high degree of both physical security and network security because of the sensitive nature of the information they contain. RAs can be placed outside of a firewall so that regular users can connect to them and can be stored in less secure locations because they do not process or contain sensitive data.

### 5.1.4. Balancing Client OCSP Requests

If a certificate is within its validity period but needs be invalidated, it can be revoked. A Certificate Manager can publish lists of revoked certificates, so that when a client needs to verify that a certificate is still valid, it can check the list. These requests are *online certificate status protocol requests*, meaning that they have a specific request and response format. The Certificate Manager has a built-in OCSP responder so that it can verify OCSP requests by itself.

However, as with certificate request traffic, a site may have a significant number of client requests to verify certificate status. Example Corp. has a large web store, and each customer's browser tries to verify the validity of their SSL certificates. Again, the CA can handle issuing the number of certificates, but the high request traffic affects its performance. In this case, Example Corp. uses the external OCSP Manager subsystem to verify certificate statuses, and the Certificate Manager only has to publish updated CRLs every so often.

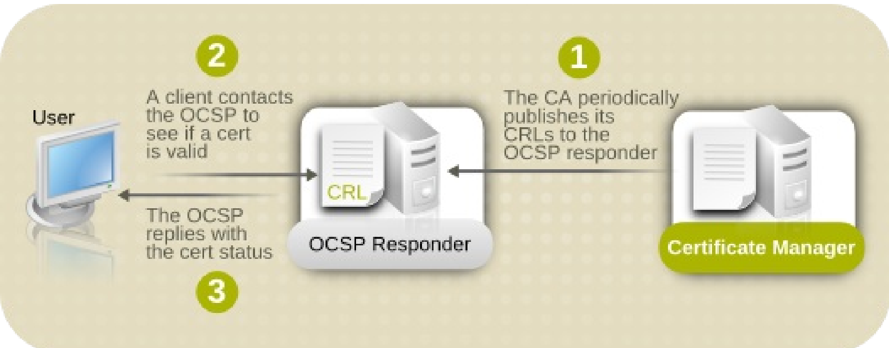



**Figure 5.4. CA and OCSP**

### 5.1.5. Planning for Smart Cards

Most certificates are enrolled through the CA. This is useful for certificates enrolled through an

application such as a web browser or web server. For managing smart cards, or tokens, Certificate System uses interlocking subsystems to create a *token management system* which handles operations for certificates and keys stored on smart cards.

Four Certificate System subsystems are involved with managing tokens:

- The Token Processing System (TPS) interacts with smart cards to help them generate and store keys and certificates for a specific entity, such as a user or device. Smart card operations go through the TPS and are forwarded to the appropriate subsystem for action, such as the Certificate Authority to generate certificates or the Data Recovery Manager to archive and recover keys.
- The Token Key Service (TKS) generates, or derives, symmetric keys used for communication between the TPS and smart card. Each set of keys generated by the TKS is unique because they are based on the card's unique ID. The keys are formatted on the smart card and are used to encrypt communications, or provide authentication, between the smart card and TPS.
- The Certificate Authority (CA) creates and revokes user certificates stored on the smart card.
- Optionally, the Data Recovery Manager (DRM) archives and recovers keys for the smart card.

The **Enterprise Security Client** is the conduit through which TPS communicates with each token over a secure HTTP channel (HTTPS), and, through the TPS, with the Certificate System.

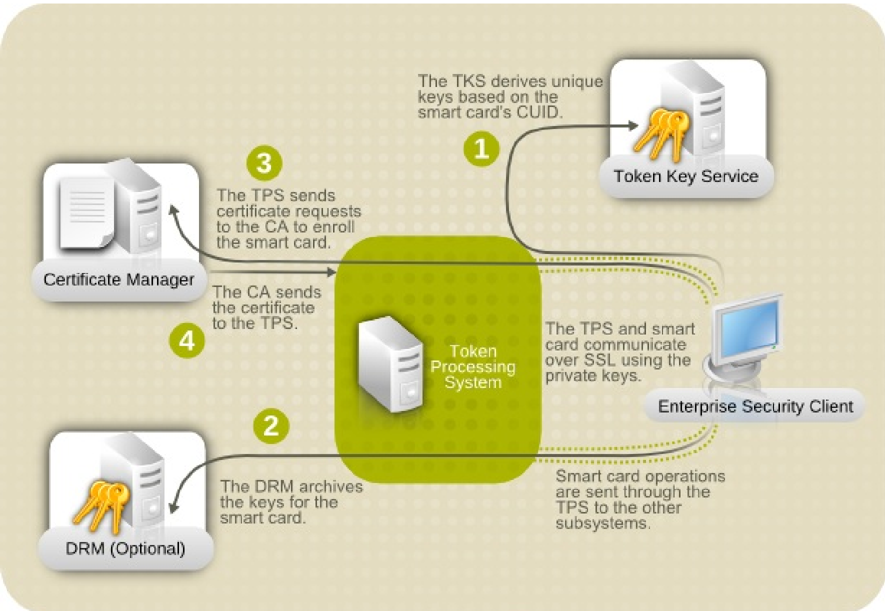
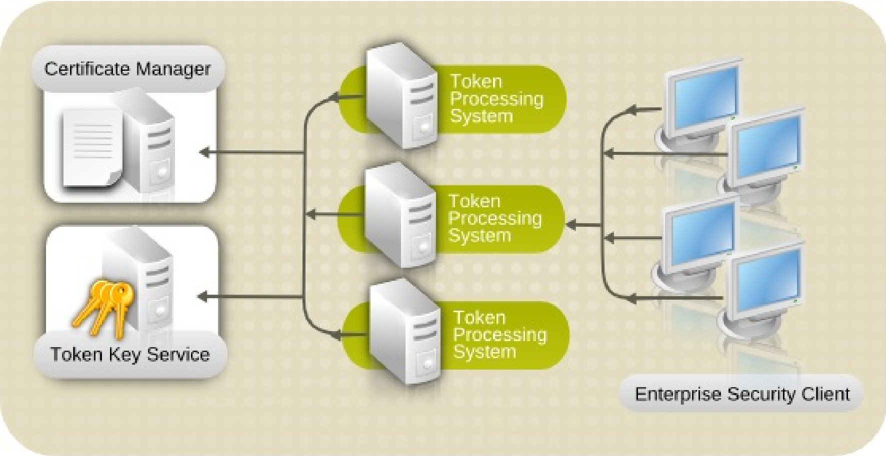



**Figure 5.5. How Certificate System Manages Smart Cards**

To use the tokens, the Token Processing System must be able to recognize and communicate with them. The tokens must first be *enrolled* to format the tokens with required keys and certificates and add the tokens to the Certificate System. The **Enterprise Security Client** provides the user interface for end entities to enroll tokens.

The token management system is very scalable. Multiple TPSs can be configured to work with a single CA, TKS, or DRM instance, while multiple Enterprise Security Clients can communicate with a single TPS. As additional clients are installed, they can point back to the TPS instance without having to reconfigure the TPS; likewise, as TPSs are added, they can point to the same CA, TKS, and DRM instances without having to reconfigure those subsystems.




After installation, the TPS configuration can be edited to use additional CA, DRM, and TKS instances for failover support, so if the primary subsystem is unavailable, the TPS can switch to the next available system without interrupting its token services.

## 5.2. Defining the Certificate Authority Hierarchy

The CA is the center of the PKI, so the relationship of CA systems, both to each other (CA hierarchy) and to other subsystems (security domain) is vital to planning a Certificate System PKI.

When there are multiple CAs in a PKI, the CAs are structured in a hierarchy or chain. The CA above another CA in a chain is called an *root CA*; a CA below another CA in the chain is called a *subordinate CA*. A CA can also be subordinate to a root outside of the Certificate System deployment; for example, a CA which functions as a root CA within the Certificate System deployment can be subordinate to a third-party CA.

A Certificate Manager (or CA) is subordinate to another CA because its CA signing certificate, the certificate that allows it to issue certificates, is issued by another CA. The CA that issued the subordinate CA signing certificate controls the CA through the contents of the CA signing certificate. The CA can constrain the subordinate CA through the kinds of certificates that it can issue, the extensions that it is allowed to include in certificates, the number of levels of subordinate CAs the subordinate CA can create, and the validity period of certificates it can issue, as well as the validity period of the subordinate CAs signing certificate.

**NOTE**

Although a subordinate CA can create certificates that violate these constraints, a client authenticating a certificate that violates those constraints will not accept that certificate.

A self-signed root CA signs its own CA signing certificate and sets its own constraints as well as setting constraints on the subordinate CA signing certificates it issues.

A Certificate Manager can be configured as either a root CA or a subordinate CA. It is easiest to make the first CA installed a self-signed root, so that it is not necessary to apply to a third party and wait for the certificate to be issued. Before deploying the full PKI, however, consider whether to have a root CA, how many to have, and where both root and subordinate CAs will be located.

### 5.2.1. Subordination to a Public CA

Chaining the Certificate System CA to a third-party public CA introduces the restrictions that public CAs place on the kinds of certificates the subordinate CA can issue and the nature of the certificate chain.

For example, a CA that chains to a third-party CA might be restricted to issuing only Secure Multipurpose Internet Mail Extensions (S/MIME) and SSL client authentication certificates, but not SSL server certificates. There are other possible restrictions with using a public CA. This may not be acceptable for some PKI deployments.

One benefit of chaining to a public CA is that the third party is responsible for submitting the root CA certificate to a web browser or other client software. This can be a major advantage for an extranet with certificates that are accessed by different companies with browsers that cannot be controlled by the administrator. Creating a root CA in the CA hierarchy means that the local organization must get the root certificate into all the browsers which will use the certificates issued by the Certificate System. There are tools to do this within an intranet, but it can be difficult to accomplish with an extranet.

### 5.2.2. Subordination to a Certificate System CA

The Certificate System CA can function as a *root CA*, meaning that the server signs its own CA signing certificate as well as other CA signing certificates, creating an organization-specific CA hierarchy. The server can alternatively be configured as a *subordinate CA*, meaning the server's CA signing key is signed by another CA in an existing CA hierarchy.

Setting up a Certificate System CA as the root CA means that the Certificate System administrator has control over all subordinate CAs by setting policies that control the contents of the CA signing certificates issued. A subordinate CA issues certificates by evaluating its own authentication and certificate profile configuration, without regard for the root CA's configuration.

### 5.2.3. Linked CA

The Certificate System Certificate Manager can function as a *linked CA*, chaining up to many third-party or public CAs for validation; this provides cross-company trust, so applications can verify certificate chains outside the company certificate hierarchy. A Certificate Manager is chained to a third-party CA by requesting the Certificate Manager's *CA signing certificate* from the third-party CA.

Related to this, the Certificate Manager also can issue *cross-pair* or *cross-signed certificates*. Cross-pair certificates create a trusted relationship between two separate CAs by issuing and storing cross-signed certificates between these two CAs. By using cross-signed certificate pairs, certificates issued outside the organization's PKI can be trusted within the system.

These are also called *bridge certificates*, related to the Federal Bridge Certification Authority (FBCA) definition.

### 5.2.4. CA Cloning

Instead of creating a hierarchy of root and subordinate CAs, it is possible to create multiple clones of a Certificate Manager and configure each clone to issue certificates within a range of serial numbers.

A *cloned* Certificate Manager uses the same CA signing key and certificate as another Certificate Manager, the *master* Certificate Manager.

Because clone CAs and original CAs use the same CA signing key and certificate to sign the certificates they issue, the *issuer name* in all the certificates is the same. Clone CAs and the original Certificate Managers issue certificates as if they are a single CA. These servers can be placed on different hosts for high availability failover support.

The advantage of cloning is that it distributes the Certificate Manager's load across several processes or even several physical machines. For a CA with a high enrollment demand, the distribution gained from cloning allows more certificates to be signed and issued in a given time interval.

A cloned Certificate Manager has the same features, such as agent and end-entity gateway functions, of

a regular Certificate Manager.

The serial numbers for clones are distributed dynamically. All of the cloned servers share a common entry which defines the next available range; when a server needs another range of numbers, it claims the range in the shared entry, and the entry increments so that a new range is available. Serial number ranges do not have to be manually assigned to the cloned Certificate Managers. This makes managing serial numbers for certificates is easy to administrate.

**TIP**

If there is a chance that a subsystem will be cloned, then it is easiest to export its key pairs during the configuration process and save them to a secure location. The key pairs for the original Certificate Manager have to be available when the clone instance is configured, so that the clone can generate its certificates from the original Certificate Manager's keys.
It is also possible to export the keys from the security databases at a later time, using the **`pk12util`** or the **`PKCS12Export`** commands.

## 5.3. Planning Security Domains and Trust Relationships

### 5.3.1. Understanding Security Domains

A *security domain* is a registry of PKI services. PKI services, such as CAs, register information about themselves in these domains so users of PKI services can find other services by inspecting the registry. The security domain service in Certificate System manages both the registration of PKI services for Certificate System subsystems and a set of shared trust policies.

The registry provides a complete view of all PKI services provided by the subsystems within that domain. Each Certificate System subsystem must be either a host or a member of a security domain.

A CA subsystem is the only subsystem which can host a security domain. The security domain shares the CA internal database for privileged user and group information to determine which users can update the security domain, register new PKI services, and issue certificates.

A security domain is created during CA configuration, which automatically creates an entry in the security domain CA's LDAP directory. Each entry contains all the important information about the domain. Every subsystem within the domain, including the CA registering the security domain, is recorded under the security domain container entry.

The URL to the CA uniquely identifies the security domain. The security domain is also given a friendly name, such as **`Example Corp Intranet PKI`**. All other subsystems — RA, DRM, TPS, TKS, OCSP, and other CAs — must become members of the security domain by supplying the security domain URL when configuring the subsystem.

Each subsystem within the security domain shares the same trust policies and trusted roots which can be retrieved from different servers and browsers. The information available in the security domain is used during configuration of a new subsystem, which makes the configuration process streamlined and automated. For example, when a TPS needs to connect to a CA, it can consult the security domain to get a list of available CAs.

Each CA has its own LDAP entry. The security domain is an organizational group underneath that CA entry:

```
ou=Security Domain,dc=server.example.com-pki-ca
```

Then there is a list of each subsystem type beneath the security domain organizational group, with a special object class (**`pkiSecurityGroup`**) to identify the group type:

```
cn=KRAList,ou=Security Domain,dc=server.example.com-pki-ca
objectClass: top
objectClass: pkiSecurityGroup
cn: KRAList
```

Each subsystem instance is then stored as a member of that group, with a special **`pkiSubsystem`** object class to identify the entry type:

```
dn: cn=drm.example.com:10444,cn=KRAList,ou=Security Domain,dc=server.example.com-
pki-ca
objectClass: top
objectClass: pkiSubsystem
cn: drm.example.com:10444
host: server.example.com
SecurePort: 10444
DomainManager: false
Clone: false
SubsystemName: Data Recovery Manager
```

If a subsystem needs to contact another subsystem to perform an operation, it contacts the CA which hosts the security domain (by invoking a servlet which connects over the administrative port of the CA). The security domain CA then retrieves the information about the subsystem from its LDAP database, and returns that information to the requesting subsystem.

The subsystem authenticates to the security domain using a subsystem certificate.

Consider the following when planning the security domain:

- The CA hosting the security domain can be signed by an external authority.
- Multiple security domains can be set up within an organization. However, each subsystem can belong to only one security domain.
- Subsystems within a domain can be cloned. Cloning subsystem instances distributes the system load and provides failover points.
- The security domain streamlines configuration between the CA and DRM; the DRM can push its KRA connector information and transport certificates automatically to the CA instead of administrators having to manually copy the certificates over to the CA.
- The Certificate System security domain allows an offline CA to be set up. In this scenario, the offline root has its own security domain. All online subordinate CAs belong to a different security domain.
- The security domain streamlines configuration between the CA and OCSP. The OCSP can push its information to the CA for the CA to set up OCSP publishing and also retrieve the CA certificate chain from the CA and store it in the internal database.

### 5.3.2. Using Trusted Managers

Trust relationships can be configured between subsystems in different security domains by using special *trusted managers*. Basically, a trusted manager is a member of a special group in the subsystem's user configuration. Another subsystem can be added as a member of that group and all of its certificates — SSL server certificate, signing certificates, storage and transport certificates (for DRMs) — can be added to establish trusted relationships.

## 5.4. Determining the Requirements for Subsystem Certificates

The CA configuration determines many of the characteristics of the certificates which it issues, regardless of the actual type of certificate being issued. Constraints on the CA's own validity period, distinguished name, and allowed encryption algorithms impact the same characteristics in their issued certificates. Additionally, the Certificate Managers have predefined profiles that set rules for different kinds of certificates that they issue, and additional profiles can be added or modified. These profile configurations also impact issued certificates.

### 5.4.1. Determining Which Certificates to Install

When a Certificate System subsystem is first installed and configured, the certificates necessary to access and administer it are automatically created. These include an agent's certificate, server certificate, and subsystem-specific certificates. These initial certificates are shown in Table 5.1, “Initial Subsystem Certificates”.

**Table 5.1. Initial Subsystem Certificates**

| Subsystem | Certificates |
|---|---|
| Certificate Manager | CA signing certificate<br>OCSP signing certificate<br>SSL server certificate<br>Subsystem certificate<br>User's (agent/administrator) certificate<br>Audit log signing certificate |
| RA | SSL server certificate<br>User's (agent/administrator) certificate |
| OCSP | OCSP signing certificate<br>SSL server certificate<br>Subsystem certificate<br>User's (agent/administrator) certificate<br>Audit log signing certificate |
| DRM | Transport certificate<br>Storage certificate<br>SSL server certificate<br>Subsystem certificate<br>User's (agent/administrator) certificate<br>Audit log signing certificate |
| TKS | SSL server certificate<br>User's (agent/administrator) certificate<br>Audit log signing certificate |
| TPS | SSL server certificate<br>User's (agent/administrator) certificate<br>Audit log signing certificate |

There are some cautionary considerations about replacing existing subsystem certificates.

- Generating new key pairs when creating a new self-signed CA certificate for a root CA will invalidate all certificates issued under the previous CA certificate.

  This means none of the certificates issued or signed by the CA using its old key will work; subordinate Certificate Managers, DRMs, OCSPs, TKSs, and TPSs will no longer function, and agents can no longer to access agent interfaces.

  This same situation occurs if a subordinate CA's CA certificate is replaced by one with a new key pair; all certificates issued by that CA are invalidated and will no longer work.

  Instead of creating new certificates from new key pairs, consider renewing the existing CA signing certificate.

- If the CA is configured to publish to the OCSP and it has a new CA signing certificate or a new CRL signing certificate, the CA must be identified again to the OCSP.

- If a new transport certificate is created for the DRM, the DRM information must be updated in the CA's configuration file, **`CS.cfg`**. The existing transport certificate must be replaced with the new one in the **`ca.connector.KRA.transportCert`** parameter.
- If a CA is cloned, then when creating a new SSL server certificate for the master Certificate Manager, the clone CAs' certificate databases all need updated with the new SSL server certificate.
- If the Certificate Manager is configured to publish certificates and CRLs to an LDAP directory and uses the SSL server certificate for SSL client authentication, then the new SSL server certificate must be requested with the appropriate extensions. After installing the certificate, the publishing directory must be configured to use the new server certificate.
- Any number of SSL server certificates can be issued for a subsystem instance, but it really only needs one SSL certificate. This certificate can be renewed or replaced as many times as necessary.

### 5.4.2. CA Distinguished Name

The core elements of a CA are a signing unit and the Certificate Manager identity. The signing unit digitally signs certificates requested by end entities. A Certificate Manager must have its own distinguished name (DN), which is listed in every certificate it issues.

Like any other certificate, a CA certificate binds a DN to a public key. A DN is a series of name-value pairs that in combination uniquely identify an entity. For example, the following DN identifies a Certificate Manager for the Engineering department of a corporation named Example Corporation:

```
cn=demoCA, o=Example Corporation, ou=Engineering, c=US
```

Many combinations of name-value pairs are possible for the Certificate Manager's DN. The DN must be unique and readily identifiable, since any end entity can examine it.

### 5.4.3. CA Signing Certificate Validity Period

Every certificate, including a Certificate Manager signing certificate, must have a validity period. The Certificate System does not restrict the validity period that can be specified. Set as long a validity period as possible, depending on the requirements for certificate renewal, the place of the CA in the certificate hierarchy, and the requirements of any public CAs that are included in the PKI.

A Certificate Manager cannot issue a certificate that has a validity period longer than the validity period of its CA signing certificate. If a request is made for a period longer than the CA certificate's validity period, the requested validity date is ignored and the CA signing certificate validity period is used.

### 5.4.4. Signing Key Type and Length

A signing key is used by a subsystem to verify and "seal" something. CAs use a CA signing certificate to sign certificates or CRLs that it issues; OCSPs use signing certificates to verify their responses to certificate status requests; all subsystems use log file signing certificates to sign their audit logs.

The signing key must be cryptographically strong to provide protection and security for its signing operations. Certificate System supports six signing algorithms, by default, two in the MD family, four in the SHA family, and one for ECC encryption:

- MD2withRSA
- MD5withRSA
- SHA1withRSA
- SHA256withRSA
- SHA512withRSA

- SHA1withEC

SHA1withRSA is the default signing algorithm for CAs for RSA certificates. SHA1withEC is the default signing algorithm for CAs for ECC certificates.

Along with a key *type*, each key has a specific *bit length*. Longer keys are considered cryptographically stronger than shorter keys. However, longer keys require more time for signing operations.

The default RSA key length in the configuration wizard is 2048 bits; for certificates that provide access to highly sensitive data or services, consider increasing the length to 4096 bits. ECC keys are much stronger than RSA keys, so the recommended length for ECC keys is 256 bits, which is equivalent in strength to a 2048-bit RSA key.

### 5.4.5. Using Certificate Extensions

An X.509 v3 certificate contains an extension field that permits any number of additional fields to be added to the certificate. Certificate extensions provide a way of adding information such as alternative subject names and usage restrictions to certificates. Older Netscape servers, such as Red Hat Directory Server and Red Hat Certificate System, require Netscape-specific extensions because they were developed before PKIX part 1 standards were defined.

The X.509 v1 certificate specification was originally designed to bind public keys to names in an X.500 directory. As certificates began to be used on the Internet and extranets and directory lookups could not always be performed, problem areas emerged that were not covered by the original specification.

- *Trust*. The X.500 specification establishes trust by means of a strict directory hierarchy. By contrast, Internet and extranet deployments frequently involve distributed trust models that do not conform to the hierarchical X.500 approach.
- *Certificate use*. Some organizations restrict how certificates are used. For example, some certificates may be restricted to client authentication only.
- *Multiple certificates*. It is not uncommon for certificate users to possess multiple certificates with identical subject names but different key material. In this case, it is necessary to identify which key and certificate should be used for what purpose.
- *Alternate names*. For some purposes, it is useful to have alternative subject names that are also bound to the public key in the certificate.
- *Additional attributes*. Some organizations store additional information in certificates, such as when it is not possible to look up information in a directory.
- *Relationship with CA*. When certificate chaining involves intermediate CAs, it is useful to have information about the relationships among CAs embedded in their certificates.
- *CRL checking*. Since it is not always possible to check a certificate's revocation status against a directory or with the original certificate authority, it is useful for certificates to include information about where to check CRLs.

The X.509 v3 specification addressed these issues by altering the certificate format to include additional information within a certificate by defining a general format for certificate extensions and specifying extensions that can be included in the certificate. The extensions defined for X.509 v3 certificates enable additional attributes to be associated with users or public keys and manage the certification hierarchy. The *Internet X.509 Public Key Infrastructure Certificate and CRL Profile* recommends a set of extensions to use for Internet certificates and standard locations for certificate or CA information. These extensions are called *standard extensions*.

> **NOTE**
>
> For more information on standard extensions, see RFC 2459, RFC 3280, and RFC 3279.

The X.509 v3 standard for certificates allows organizations to define custom extensions and include them in certificates. These extensions are called *private*, *proprietary*, or *custom* extensions, and they carry information unique to an organization or business. Applications may not able to validate certificates that contain private critical extensions, so it not recommended that these be used in wide-spread situations.

Before the X.509 v3 standard was finalized, Netscape and other companies had to address some of the most pressing issues with their own extension definitions. For example, applications such as Netscape Navigator and Enterprise Server supported an extension known as the *Netscape Certificate Type Extension* that specifies the type of certificate issued, such as client, server, or email. To maintain compatibility with older versions of browsers that were released before the X.509 v3 specification was finalized, certain kinds of certificates should include some of these Netscape extensions.

The X.500 and X.509 specifications are controlled by the International Telecommunication Union (ITU), an international organization that primarily serves large telecommunication companies, government organizations, and other entities concerned with the international telecommunications network. The Internet Engineering Task Force (IETF), which controls many of the standards that underlie the Internet, is currently developing public-key infrastructure X.509 (PKIX) standards. These proposed standards further refine the X.509 v3 approach to extensions for use on the Internet. The recommendations for certificates and CRLs have reached proposed standard status and are in a document referred to as *PKIX Part 1*.

Two other standards, Abstract Syntax Notation One (ASN.1) and Distinguished Encoding Rules (DER), are used with Certificate System and certificates in general. These are specified in the CCITT Recommendations X.208 and X.209. For a quick summary of ASN.1 and DER, see *A Layman's Guide to a Subset of ASN.1, BER, and DER*, which is available at RSA Laboratories' web site, http://www.rsa.com.

#### 5.4.5.1. Structure of Certificate Extensions

In RFC 3280, an X.509 certificate extension is defined as follows:

```
Extension  ::=  SEQUENCE  {

extnID OBJECT IDENTIFIER,
critical BOOLEAN DEFAULT FALSE,
extnValue OCTET STRING  }
```

The means a certificate extension consists of the following:

- The object identifier (OID) for the extension. This identifier uniquely identifies the extension. It also determines the ASN.1 type of value in the value field and how the value is interpreted. When an extension appears in a certificate, the OID appears as the extension ID field (**`extnID`**) and the corresponding ASN.1 encoded structure appears as the value of the octet string (**`extnValue`**).
- A flag or Boolean field called **`critical`**.

  The value, which can be either **`true`** or **`false`**, assigned to this field indicates whether the extension is critical or noncritical to the certificate.

  - If the extension is critical and the certificate is sent to an application that does not understand the extension based on the extension's ID, the application must reject the certificate.

  - If the extension is not critical and the certificate is sent to an application that does not understand the extension based on the extension's ID, the application can ignore the extension and accept the certificate.
- An octet string containing the DER encoding of the value of the extension.

Typically, the application receiving the certificate checks the extension ID to determine if it can recognize the ID. If it can, it uses the extension ID to determine the type of value used.

Some of the standard extensions defined in the X.509 v3 standard include the following:

- Authority Key Identifier extension, which identifies the CA's public key, the key used to sign the certificate.
- Subject Key Identifier extension, which identifies the subject's public key, the key being certified.

**NOTE**

Not all applications support certificates with version 3 extensions. Applications that do support these extensions may not be able to interpret some or all of these specific extensions.

### 5.4.6. Using and Customizing Certificate Profiles

Certificates have different types and different applications. They can be used to establish a single sign-on environment for a corporate network, to set up VPNs, to encrypt email, or to authenticate to a website. The requirements for all of these certificates can be different, just as there may also be different requirements for the same type of certificate for different kinds of users. These certificate characteristics are set in *certificate profiles*. The Certificate Manager defines a set of certificate profiles that it uses as enrollment forms when users or machines request certificates.

A certificate profile defines everything associated with issuing a particular type of certificate, including the authentication method, the certificate content (defaults), constraints for the values of the content, and the contents of the input and output for the certificate profile. Enrollment requests are submitted to a certificate profile and are then subject to the defaults and constraints set in that certificate profile. These constraints are in place whether the request is submitted through the input form associated with the certificate profile or through other means. The certificate that is issued from a certificate profile request contains the content required by the defaults with the information required by the default parameters. The constraints provide rules for what content is allowed in the certificate.

For example, a certificate profile for user certificates defines all aspects of that certificate, including the validity period of the certificate. The default validity period can be set to two years, and a constraint can be set on the profile that the validity period for certificates requested through this certificate profile cannot exceed two years. When a user requests a certificate using the input form associated with this certificate profile, the issued certificate contains the information specified in the defaults and will be valid for two years. If the user submits a pre-formatted request for a certificate with a validity period of four years, the request is rejected since the constraints allow a maximum of two years validity period for this type of certificate.

A set of certificate profiles have been predefined for the most common certificates issued. These certificate profiles define defaults and constraints, associate the authentication method, and define the needed inputs and outputs for the certificate profile.

The parameters of the default certificate profiles - the authentication method, the defaults, the constraints used in each profile, the values assigned to any of the parameters in a profile, the input, and the output - can be modified. It is also possible to create new certificate profiles for other types of

certificates or for creating more than one certificate profile for a certificate type. There can be multiple certificate profiles for a particular type of certificate to issue the same type of certificate with a different authentication method or different definitions for the defaults and constraints. For example, there can be two certificate profiles for enrollment of SSL server certificates where one certificate profile issues certificates with a validity period of six months and another certificate profile issues certificates with a validity period of two years.

An input sets a text field in the enrollment form and what kind of information needs gathered from the end entity; this includes setting the text area for a certificate request to be pasted, which allows a request to be created outside the input form with any of the request information desired. The input values are set as values in the certificate. The default inputs are not configurable in the Certificate System.

An output specifies how the response page to a successful enrollment is presented. It usually displays the certificate in a user-readable format. The default output shows a printable version of the resultant certificate; other outputs set the type of information generated at the end of the enrollment, such as PKCS #7.

Policy sets are sets of constraints and default extensions attached to every certificate processed through the profile. The extensions define certificate content such as validity periods and subject name requirements. A profile handles one certificate request, but a single request can contain information for multiple certificates. A PKCS#10 request contains a single public key. One CRMF request can contain multiple public keys, meaning multiple certificate requests. A profile may contain multiple sets of policies, with each set specifying how to handle one certificate request within a CRMF request.

An administrator sets up a certificate profile by associating an existing authentication plug-in, or method, with the certificate profile; enabling and configuring defaults and constraints; and defining inputs and outputs. The administrator can use the existing certificate profiles, modify the existing certificate profiles, create new certificate profiles, and delete any certificate profile that will not be used in this PKI.

Once a certificate profile is set up, it appears on the **`Manage Certificate Profiles`** page of the agent services page where an agent can approve, and thus enable, a certificate profile. Once the certificate profile is enabled, it appears on the **`Certificate Profile`** tab of the end-entities page where end entities can enroll for a certificate using the certificate profile.

The certificate profile enrollment page in the end-entities interface contains links to each certificate profile that has been enabled by the agents. When an end entity selects one of those links, an enrollment page appears containing an enrollment form specific to that certificate profile. The enrollment page is dynamically generated from the inputs defined for the profile. If an authentication plug-in is configured, additional fields may be added to authenticate the user.

When an end entity submits a certificate profile request that is associated with an agent-approved (manual) enrollment, an enrollment where no authentication plug-in is configured, the certificate request is queued in the agent services interface. The agent can change some aspects of the enrollment, request, validate it, cancel it, reject it, update it, or approve it. The agent is able to update the request without submitting it or validate that the request adheres to the profile's defaults and constraints. This validation procedure is only for verification and does not result in the request being submitted. The agent is bound by the constraints set; they cannot change the request in such a way that a constraint is violated. The signed approval is immediately processed, and a certificate is issued.

When a certificate profile is associated with an authentication method, the request is approved immediately and generates a certificate automatically if the user successfully authenticates, all the information required is provided, and the request does not violate any of the constraints set up for the certificate profile. There are profile policies which allow user-supplied settings like subject names or validity periods. The certificate profile framework can also preserve user-defined content set in the

original certificate request in the issued certificate.

The issued certificate contains the content defined in the defaults for this certificate profile, such as the extensions and validity period for the certificate. The content of the certificate is constrained by the constraints set for each default. Multiple policies (defaults and constraints) can be set for one profile, distinguishing each set by using the same value in the policy set ID. This is particularly useful for dealing with dual keys enrollment where encryption keys and signing keys are submitted to the same profile. The server evaluates each set with each request it receives. When a single certificate is issued, one set is evaluated, and any other sets are ignored. When dual-key pairs are issued, the first set is evaluated with the first certificate request, and the second set is evaluated with the second certificate request. There is no need for more than one set for issuing a single certificate or more than two sets for issuing dual-key pairs.

Tailor the profiles for the organization to the real needs and anticipated certificate types used by the organization:

- Decide which certificate profiles are needed in the PKI. There should be at least one profile for each type of certificate issued. There can be more than one certificate profile for each type of certificate to set different authentication methods or different defaults and constraints for a particular type of certificate type. Any certificate profile available in the administrative interface can be approved by an agent and then used by an end entity to enroll.
- Delete any certificate profiles that will not be used.
- Modify the existing certificate profiles for specific characteristics for the company's certificates.
  - Change the defaults set up in the certificate profile, the values of the parameters set in the defaults, or the constraints that control the certificate content.
  - Change the constraints set up by changing the value of the parameters.
  - Change the authentication method.
  - Change the inputs by adding or deleting inputs in the certificate profile, which control the fields on the input page.
  - Add or delete the output.

### 5.4.7. Planning Authentication Methods

As implied in Section 5.4.6, “Using and Customizing Certificate Profiles”, *authentication* for the certificate process means the way that a user or entity requesting a certificate proves that they are who they say they are. There are three ways that the Certificate System can authenticate an entity:

- In *agent-approved* enrollment, end-entity requests are sent to an agent for approval. The agent approves the certificate request.
- In *automatic* enrollment, end-entity requests are authenticated using a plug-in, and then the certificate request is processed; an agent is not involved in the enrollment process.
- In *CMC enrollment*, a third party application can create a request that is signed by an agent and then automatically processed.

A Certificate Manager is initially configured for agent-approved enrollment and for CMC authentication. Automated enrollment is enabled by configuring one of the authentication plug-in modules. More than one authentication method can be configured in a single instance of a subsystem. The HTML registration pages contain hidden values specifying the method used. With certificate profiles, the end-entity enrollment pages are dynamically-generated for each enabled profile. The authentication method associated with this certificate profile is specified in the dynamically-generated enrollment page.

The authentication process is simple.

1. An end entity submits a request for enrollment. The form used to submit the request identifies the

method of authentication and enrollment. All HTML forms are dynamically-generated by the profiles, which automatically associate the appropriate authentication method with the form.

2. If the authentication method is an agent-approved enrollment, the request is sent to the request queue of the CA agent. If the automated notification for a request in queue is set, an email is sent to the appropriate agent that a new request has been received. The agent can modify the request as allowed for that form and the profile constraints. Once approved, the request must pass the certificate profiles set for the Certificate Manager, and then the certificate is issued. When the certificate is issued, it is stored in the internal database and can be retrieved by the end entity from the end-entities page by serial number or by request ID.
3. If the authentication method is automated, the end entity submits the request along with required information to authenticate the user, such as an LDAP username and password. When the user is successfully authenticated, the request is processed without being sent to an agent's queue. If the request passes the certificate profile configuration of the Certificate Manager, the certificate is issued and stored in the internal database. It is delivered to the end entity immediately through the HTML forms.

The requirements for how a certificate request is authenticated can have a direct impact on the necessary subsystems and profile settings. For example, if an agent-approved enrollment requires that an agent meet the requester in person and verify their identity through supported documentation, the authentication process can be time-intensive, as well as constrained by the physical availability of both the agent and the requester. In that case, having numerous local RAs may be preferable to centralized CAs.

### 5.4.8. Publishing Certificates and CRLs

A CA can publish both certificates and CRLs. Certificates can be published to a plain file or to an LDAP directory; CRLs can be published to file or an LDAP directory, as well, and can also be published to an OCSP responder to handle certificate verification.

Configuring publishing is fairly straightforward and is easily adjusted. For continuity and accessibility, though, it is good to plan out where certificates and CRLs need to be published and what clients need to be able to access them.

Publishing to an LDAP directory requires special configuration in the directory for publishing to work:

- If certificates are published to the directory, than every user or server to which a certificate is issued must have a corresponding entry in the LDAP directory.
- If CRLs are published to the directory, than they must be published to an entry for the CA which issued them.
- For SSL, the directory service has to be configured in SSL and, optionally, be configured to allow the Certificate Manager to use certificate-based authentication.
- The directory administrator should configure appropriate access control rules to control DN (entry name) and password based authentication to the LDAP directory.

### 5.4.9. Renewing or Reissuing CA Signing Certificates

When a CA signing certificate expires, all certificates signed with the CA's corresponding signing key become invalid. End entities use information in the CA certificate to verify the certificate's authenticity. If the CA certificate itself has expired, applications cannot chain the certificate to a trusted CA.

There are two ways of resolving CA certificate expiration:

- *Renewing* a CA certificate involves issuing a new CA certificate with the same subject name and public and private key material as the old CA certificate, but with an extended validity period. As long

as the new CA certificate is distributed to all users before the old CA certificate expires, renewing the certificate allows certificates issued under the old CA certificate to continue working for the full duration of their validity periods.

- *Reissuing* a CA certificate involves issuing a new CA certificate with a new name, public and private key material, and validity period. This avoids some problems associated with renewing a CA certificate, but it requires more work for both administrators and users to implement. All certificates issued by the old CA, including those that have not yet expired, must be renewed by the new CA.

There are problems and advantages with either renewing or reissuing a CA certificate. Begin planning the CA certificate renewal or re-issuance before installing any Certificate Managers, and consider the ramifications the planned procedures may have for extensions, policies, and other aspects of the PKI deployment.

> **NOTE**
>
> Correct use of extensions, for example the **`authorityKeyIdentifier`** extension, can affect the transition from an old CA certificate to a new one.

## 5.5. Planning for Network and Physical Security

When deploying any Certificate System subsystem, the physical and network security of the subsystem instance has to be considered because of the sensitivity of the data generated and stored by the subsystems.

### 5.5.1. Considering Firewalls

There are two considerations about using firewalls with Certificate System subsystems:

- Protecting sensitive subsystems from unauthorized access
- Allowing appropriate access to other subsystems and clients outside of the firewall

The CA, DRM, and TKS are always placed inside a firewall because they contain critical information that can cause devastating security consequences if they are compromised.

The RA is frequently placed outside the firewall and the TPS and OCSP can be. Likewise, other services and clients used by the Certificate System can be on a different machine outside the firewall. In that case, the local networks have to be configured to allow access between the subsystems behind the firewall and the services outside it.

The LDAP database can be on a different server, even on a different network, than the subsystem which uses it. In this case, all LDAP ports (**`389`** for LDAP and **`636`** for LDAPS, by default) need to be open in the firewall to allow traffic to the directory service. Without access to the LDAP database, all subsystem operations can fail.

As part of configuring the firewalls, if iptables is enabled, then it must have configured policies to allow communication over the appropriate Certificate System ports. Configuring iptables is described in the Red Hat Enterprise Linux *Deployment Guide*, such as "Using iptables."

### 5.5.2. Considering Physical Security and Location

Because of the sensitive data they hold, consider keeping the CA, DRM, and TKS in a secure facility with adequate physical access restrictions. Just as network access to the systems needs to be restricted, the physical access also needs to be restricted.

Along with finding a secure location, consider the proximity to the subsystem agents and administrators. Key recovery, for example, can require multiple agents to give approval; if these agents are spread out over a large geographical area, then the time differences may negatively impact the ability to retrieve keys. Plan the physical locations of the subsystems, then according to the locations of the agents and administrators who will manage the subsystem.

### 5.5.3. Port Considerations

The default subsystems are installed and configured with a separate TCP/IP port for each service, called *port separation*. Each Certificate System subsystem uses up to five ports:

- A standard port
- An SSL port for end users services
- An SSL port for agent services
- An SSL port for the administrative console or admin services
- A web server port (Tomcat for CA, DRM, OCSP, and TKS subsystems, Apache for the TPS and RA subsystems)

All of the service pages and interfaces described in Section 2.2, “Red Hat Certificate System Services” are connected to using the instance's URL and the corresponding port number. For example:

```
https://server.example.com:9444/ca/ee/ca
```

To access the admin console, the URL specifies the admin port:

```
pkiconsole https://server.example.com:9445/ca
```

All agent and admin functions require SSL client authentication. For requests from end entities, the Certificate System listens on both the SSL (encrypted) port and non-SSL ports.

The ports for the different services to use are defined in the **server.xml** file for the CA, OCSP, DRM, and TKS and in the **httpd.conf** and **nss.conf** files for the RA and TPS. If a port is not used, it can be disabled in that file. For example:

```
<Service name="Catalina">
    <!--Connector port="9180" ... /-->  unused standard port
    <Connector port="9443" ... />
```

The default ports for the first instance installed on a machine are already defined, but it is possible to choose ports different ports for additional instances. Whenever a new instance in installed, it can be configured to have a single SSL port or to use three separate SSL ports for the different interface. Whichever way you choose, make sure that the new ports are unique on the host system.

To verify that a port is available for use, check the appropriate file for the operating system. Port numbers for network-accessible services are usually maintained in a file named **services**. On Red Hat Enterprise Linux, it is also helpful to confirm that a port is not assigned by SELinux, by running the command **semanage port -l** to list all ports which currently have an SELinux context.

When a new subsystem instance is created, any number between 1 and 65535 can be specified as the secure port number.

For additional instances, created with the **pkicreate** utility, it is possible to configure the instance with port separation or using a single, consolidated SSL port for all services. Port separation is more secure, but it depends on the maintenance costs, users, and network which is preferable.

**Table 5.2. Default Port Assignments for Certificate System 8.0**

| Subsystem | Standard | End-Entity SSL | Agent SSL | Admin SSL | Tomcat |
|---|---|---|---|---|---|
| CA | 9180 | 9444 | 9443 | 9445 | 9701 |
| RA | 12888 | | 12889 | 12889 | |
| OCSP | 11180 | 11444 | 11443 | 11445 | 11701 |
| DRM | 10180 | 10444 | 10443 | 10445 | 10701 |
| TKS | 13180 | 13444 | 13443 | 13445 | 13701 |
| TPS | 7888 | | 7889 | 7889 | |

## 5.6. Tokens for Storing Certificate System Subsystem Keys and Certificates

A *token* is a hardware or software device that performs cryptographic functions and stores public-key certificates, cryptographic keys, and other data. The Certificate System defines two types of tokens, *internal* and *external*, for storing key pairs and certificates that belong to the Certificate System subsystems.

An internal (software) token is a pair of files, usually called the *certificate database* (**`cert8.db`**) and *key database* (**`key3.db`**), that the Certificate System uses to generate and store its key pairs and certificates. The Certificate System automatically generates these files in the filesystem of its host machine when first using the internal token. These files are created during the Certificate System subsystem configuration if the internal token was selected for key-pair generation.

These security databases are located in the **`/var/lib/`*`subsystem_name`*`/alias`** directory.

An external token refers to an external hardware device, such as a smart card or hardware security module (HSM), that the Certificate System uses to generate and store its key pairs and certificates. The Certificate System supports any hardware tokens that are compliant with PKCS #11.

PKCS #11 is a standard set of APIs and shared libraries which isolate an application from the details of the cryptographic device. This enables the application to provide a unified interface for PKCS #11-compliant cryptographic devices.

The PKCS #11 module implemented in the Certificate System supports cryptographic devices supplied by many different manufacturers. This module allows the Certificate System to plug in shared libraries supplied by manufacturers of external encryption devices and use them for generating and storing keys and certificates for the Certificate System managers.

Consider using external tokens for generating and storing the key pairs and certificates used by Certificate System. These devices are another security measure to safeguard private keys because hardware tokens are sometimes considered more secure than software tokens.

Before using external tokens, plan how the external token is going to be used with the subsystem:

- All system keys for a subsystem must be generated on the same token.
- The subsystem must be installed in an empty HSM slot. If the HSM slot has previously been used to store other keys, then use the HSM vendor's utilities to delete the contents of the slot. The Certificate System has to be able to create certificates and keys on the slot with default nicknames. If not

properly cleaned up, the names of these objects may collide with previous instances.

The Certificate System can also use *hardware cryptographic accelerators* with external tokens. Many of the accelerators provide the following security features:

- Fast SSL connections. Speed is important to accommodate a high number of simultaneous enrollment or service requests.
- Hardware protection of private keys. These devices behave like smart cards by not allowing private keys to be copied or removed from the hardware token. This is important as a precaution against key theft from an active attack of an online Certificate Manager.

The Certificate System supports the nCipher netHSM hardware security module (HSM), by default. Certificate System-supported HSMs are automatically added to the **`secmod.db`** database with **`modutil`** during the pre-configuration stage of the installation, if the PKCS #11 library modules are in the default installation paths.

During configuration, the **`Security Modules`** panel displays the supported modules, along with the NSS internal software PKCS #11 module. All supported modules that are detected show a status of **`Found`** and is individually marked as either **`Logged in`** or **`Not logged in`**. If a token is found but not logged in, it is possible to log in using the **`Login`** under **`Operations`**. If the administrator can log into a token successfully, the password is stored in a configuration file. At the next start or restart of the Certificate System instance, the passwords in the password store are used to attempt a login for each corresponding token.

Administrators are allowed to select any of the tokens that are logged in as the default token, which is used to generate system keys.

## 5.7. Questions for Planning the Certificate System

- Will the PKI allow replacement keys? Will it require key archival and recovery?
- Will local offices need to process their own certificate requests? Will there be a large number of certificate requests?
- Will many external clients need to validate certificate status? Can the internal OCSP in the Certificate Manager handle the load?
- Will the organization use smart cards? If so, will temporary smart cards be allowed if smart cards are mislaid, requiring key archival and recovery?
- What are the requirements for the CA signing certificate? Does the Certificate System need control over attributes like the validity period? How will the CA certificates be distributed?
- Will any subsystems need to be cloned and, if so, what are the methods for securely storing their key materials?
- How many security domains will be created, and what subsystem instances will be placed in each domain?
- Are trusted relationships required for subsystems in different security domains?
- What kinds of certificates will be issued? What characteristics do they need to have, and what profile settings are available for those characteristics? What restrictions need to be placed on the certificates?
- What are the requirements for approving a certificate request? How does the requester authenticate himself, and what kind of process is required to approve the request?
- Where will certificates and CRLs be published? What configuration needs to be done on the receiving end for publishing to work? What kinds of certificates or CRLs need to be published and how frequently?

- What subsystems should be placed behind firewalls? What clients or other subsystems need to access those firewall-protected subsystems and how will access be granted? Is firewall access allowed for the LDAP database?
- What subsystems need to be physically secured? How will access be granted, and who will be granted access?
- What is the physical location of all agents and administrators? What is the physical location of the subsystems? How will administrators or agents access the subsystem services in a timely-manner? Is it necessary to have subsystems in each geographical location or time zone?
- What ports should be assigned for each subsystem? Is it necessary to have a single SSL port, or is it better to have port separation for extra security?

# Glossary

## A

**access control**

The process of controlling what particular users are allowed to do. For example, access control to servers is typically based on an identity, established by a password or a certificate, and on rules regarding what that entity can do. See also access control list (ACL).

**access control instructions (ACI)**

An access rule that specifies how subjects requesting access are to be identified or what rights are allowed or denied for a particular subject. See access control list (ACL).

**access control list (ACL)**

A collection of access control entries that define a hierarchy of access rules to be evaluated when a server receives a request for access to a particular resource. See access control instructions (ACI).

**administrator**

The person who installs and configures one or more Certificate System managers and sets up privileged users, or agents, for them. See also agent.

**agent**

A user who belongs to a group authorized to manage agent services for a Certificate System manager. See also Certificate Manager agent, Data Recovery Manager agent.

**agent services**

1. Services that can be administered by a Certificate System agent through HTML pages served by the Certificate System subsystem for which the agent has been assigned the necessary privileges.

2. The HTML pages for administering such services.

**agent-approved enrollment**

An enrollment that requires an agent to approve the request before the certificate is issued.

### attribute value assertion (AVA)

An assertion of the form *attribute* = *value*, where *attribute* is a tag, such as **`o`** (organization) or **`uid`** (user ID), and *value* is a value such as "Red Hat, Inc." or a login name. AVAs are used to form the distinguished name (DN) that identifies the subject of a certificate, called the subject name of the certificate.

### audit log

A log that records various system events. This log can be signed, providing proof that it was not tampered with, and can only be read by an auditor user.

### auditor

A privileged user who can view the signed audit logs.

### authentication

Confident identification; assurance that a party to some computerized transaction is not an impostor. Authentication typically involves the use of a password, certificate, PIN, or other information to validate identity over a computer network. See also password-based authentication, certificate-based authentication, client authentication, server authentication.

### authentication module

A set of rules (implemented as a Java™ class) for authenticating an end entity, agent, administrator, or any other entity that needs to interact with a Certificate System subsystem. In the case of typical end-user enrollment, after the user has supplied the information requested by the enrollment form, the enrollment servlet uses an authentication module associated with that form to validate the information and authenticate the user's identity. See servlet.

### authorization

Permission to access a resource controlled by a server. Authorization typically takes place after the ACLs associated with a resource have been evaluated by a server. See access control list (ACL).

### automated enrollment

A way of configuring a Certificate System subsystem that allows automatic authentication for end-entity enrollment, without human intervention. With this form of authentication, a certificate request that completes authentication module processing successfully is automatically approved for profile processing and certificate issuance.

## B

### bind DN

A user ID, in the form of a distinguished name (DN), used with a password to authenticate to Red Hat Directory Server.

## C

### CA certificate

A certificate that identifies a certificate authority. See also certificate authority (CA), subordinate CA, root CA.

### CA hierarchy

A hierarchy of CAs in which a root CA delegates the authority to issue certificates to subordinate CAs. Subordinate CAs can also expand the hierarchy by delegating issuing status to other CAs. See also certificate authority (CA), subordinate CA, root CA.

### CA server key

The SSL server key of the server providing a CA service.

### CA signing key

The private key that corresponds to the public key in the CA certificate. A CA uses its signing key to sign certificates and CRLs.

### certificate

Digital data, formatted according to the X.509 standard, that specifies the name of an individual, company, or other entity (the subject name of the certificate) and certifies that a public key, which is also included in the certificate, belongs to that entity. A certificate is issued and digitally signed by a certificate authority (CA). A certificate's validity can be verified by checking the CA's digital signature through public-key cryptography techniques. To be trusted within a public-key infrastructure (PKI), a certificate must be issued and signed by a CA that is trusted by other entities enrolled in the PKI.

### certificate authority (CA)

A trusted entity that issues a certificate after verifying the identity of the person or entity the certificate is intended to identify. A CA also renews and revokes certificates and generates CRLs. The entity named in the issuer field of a certificate is always a CA. Certificate authorities can be independent third parties or a person or organization using certificate-issuing server software, such as Red Hat Certificate System.

### certificate chain

A hierarchical series of certificates signed by successive certificate authorities. A CA certificate identifies a certificate authority (CA) and is used to sign certificates issued by that authority. A CA certificate can in turn be signed by the CA certificate of a parent CA, and so on up to a root CA. Certificate System allows any end entity to retrieve all the certificates in a certificate chain.

### certificate extensions

An X.509 v3 certificate contains an extensions field that permits any number of additional fields to be added to the certificate. Certificate extensions provide a way of adding information such as alternative subject names and usage restrictions to certificates. A number of standard extensions have been defined by the PKIX working group.

### certificate fingerprint

A one-way hash associated with a certificate. The number is not part of the certificate itself, but is produced by applying a hash function to the contents of the certificate. If the contents of the certificate changes, even by a single character, the same function produces a different number. Certificate fingerprints can therefore be used to verify that certificates have not been tampered with.

### Certificate Management Message Formats (CMMF)

Message formats used to convey certificate requests and revocation requests from end entities to a Certificate Manager and to send a variety of information to end entities. A proposed standard from the Internet Engineering Task Force (IETF) PKIX working group. CMMF is subsumed by another proposed standard, Certificate Management Messages over Cryptographic Message Syntax (CMC). For detailed information, see http://www.ietf.org/internet-drafts/draft-ietf-pkix-cmmf-02.txt.

### Certificate Management Messages over Cryptographic Message Syntax (CMC)

Message format used to convey a request for a certificate to a Certificate Manager. A proposed standard from the Internet Engineering Task Force (IETF) PKIX working group. For detailed information, see http://www.ietf.org/internet-drafts/draft-ietf-pkix-cmc-02.txt.

### Certificate Manager

An independent Certificate System subsystem that acts as a certificate authority. A Certificate Manager instance issues, renews, and revokes certificates, which it can publish along with CRLs to an LDAP directory. It accepts requests from end entities. See certificate authority (CA).

### Certificate Manager agent

A user who belongs to a group authorized to manage agent services for a Certificate Manager. These services include the ability to access and modify (approve and reject) certificate requests and issue certificates.

### certificate profile

A set of configuration settings that defines a certain type of enrollment. The certificate profile sets policies for a particular type of enrollment along with an authentication method in a certificate profile.

### Certificate Request Message Format (CRMF)

Format used for messages related to management of X.509 certificates. This format is a subset of CMMF. See also Certificate Management Message Formats (CMMF). For detailed information, see ftp://ftp.isi.edu/in-notes/rfc2511.txt.

### certificate revocation list (CRL)

As defined by the X.509 standard, a list of revoked certificates by serial number, generated and signed by a certificate authority (CA).

**Certificate System**

See Red Hat Certificate System, Cryptographic Message Syntax (CS).

**Certificate System console**

A console that can be opened for any single Certificate System instance. A Certificate System console allows the Certificate System administrator to control configuration settings for the corresponding Certificate System instance.

**Certificate System instance**

An instance of a Certificate System subsystem, comprising both code and data and treated as a discrete entity.

**Certificate System subsystem**

One of the five Certificate System managers: Certificate Manager, Online Certificate Status Manager, Data Recovery Manager, Token Key Service, or Token Processing System.

**certificate-based authentication**

Authentication based on certificates and public-key cryptography. See also password-based authentication.

**chain of trust**

See certificate chain.

**chained CA**

See linked CA.

**cipher**

See cryptographic algorithm.

**client authentication**

The process of identifying a client to a server, such as with a name and password or with a certificate and some digitally signed data. See certificate-based authentication, password-based authentication, server authentication.

**client SSL certificate**

A certificate used to identify a client to a server using the SSL protocol. See Secure Sockets Layer (SSL).

**CMC**

See Certificate Management Messages over Cryptographic Message Syntax (CMC).

### CMC Enrollment

Features that allow either signed enrollment or signed revocation requests to be sent to a Certificate Manager using an agent's signing certificate. These requests are then automatically processed by the Certificate Manager.

### CMMF

See Certificate Management Message Formats (CMMF).

### CRL

See certificate revocation list (CRL).

### CRMF

See Certificate Request Message Format (CRMF).

### cross-certification

The exchange of certificates by two CAs in different certification hierarchies, or chains. Cross-certification extends the chain of trust so that it encompasses both hierarchies. See also certificate authority (CA).

### cross-pair certificate

A certificate issued by one CA to another CA which is then stored by both CAs to form a circle of trust. The two CAs issue certificates to each other, and then store both cross-pair certificates as a certificate pair.

### cryptographic algorithm

A set of rules or directions used to perform cryptographic operations such as encryption and decryption.

### Cryptographic Message Syntax (CS)

The syntax used to digitally sign, digest, authenticate, or encrypt arbitrary messages, such as CMMF.

### cryptographic module

See PKCS #11 module.

### cryptographic service provider (CSP)

A cryptographic module that performs cryptographic services, such as key generation, key storage, and encryption, on behalf of software that uses a standard interface such as that defined by PKCS #11 to request such services.

**CSP**

See cryptographic service provider (CSP).

## D

**Data Encryption Standard (DES)**

A FIPS-approved cryptographic algorithm required by FIPS 140-1 and specified by FIPS PUBS 46-2. DES, which uses 56-bit keys, is a standard encryption and decryption algorithm that has been used successfully throughout the world for more than 20 years. See also FIPS PUBS 140-1. For detailed information, see http://www.itl.nist.gov/div897/pubs/fip46-2.htm

**Data Recovery Manager**

An optional, independent Certificate System subsystem that manages the long-term archival and recovery of RSA encryption keys for end entities. A Certificate Manager can be configured to archive end entities' encryption keys with a Data Recovery Manager before issuing new certificates. The Data Recovery Manager is useful only if end entities are encrypting data, such as sensitive email, that the organization may need to recover someday. It can be used only with end entities that support dual key pairs: two separate key pairs, one for encryption and one for digital signatures.

**Data Recovery Manager agent**

A user who belongs to a group authorized to manage agent services for a Data Recovery Manager, including managing the request queue and authorizing recovery operation using HTML-based administration pages.

**Data Recovery Manager recovery agent**

One of the *m of n* people who own portions of the storage key for the Data Recovery Manager.

**Data Recovery Manager storage key**

Special key used by the Data Recovery Manager to encrypt the end entity's encryption key after it has been decrypted with the Data Recovery Manager's private transport key. The storage key never leaves the Data Recovery Manager.

**Data Recovery Manager transport certificate**

Certifies the public key used by an end entity to encrypt the entity's encryption key for transport to the Data Recovery Manager. The Data Recovery Manager uses the private key corresponding to the certified public key to decrypt the end entity's key before encrypting it with the storage key.

**decryption**

Unscrambling data that has been encrypted. See encryption.

**delta CRL**

A CRL containing a list of those certificates that have been revoked since the last full CRL was

issued.

### digital ID

See certificate.

### digital signature

To create a digital signature, the signing software first creates a one-way hash from the data to be signed, such as a newly issued certificate. The one-way hash is then encrypted with the private key of the signer. The resulting digital signature is unique for each piece of data signed. Even a single comma added to a message changes the digital signature for that message. Successful decryption of the digital signature with the signer's public key and comparison with another hash of the same data provides tamper detection. Verification of the certificate chain for the certificate containing the public key provides authentication of the signer. See also nonrepudiation, encryption.

### distinguished name (DN)

A series of AVAs that identify the subject of a certificate. See attribute value assertion (AVA).

### distribution points

Used for CRLs to define a set of certificates. Each distribution point is defined by a set of certificates that are issued. A CRL can be created for a particular distribution point.

### dual key pair

Two public-private key pairs, four keys altogether, corresponding to two separate certificates. The private key of one pair is used for signing operations, and the public and private keys of the other pair are used for encryption and decryption operations. Each pair corresponds to a separate certificate. See also encryption key, public-key cryptography, signing key.

## E

### eavesdropping

Surreptitious interception of information sent over a network by an entity for which the information is not intended.

### Elliptic Curve Cryptography (ECC)

A cryptographic algorithm which uses elliptic curves to create additive logarithms for the mathematical problems which are the basis of the cryptographic keys. ECC ciphers are more efficient to use than RSA ciphers and, because of their intrinsic complexity, are stronger at smaller bits than RSA ciphers. For more information, see http://ietfreport.isoc.org/idref/draft-ietf-tls-ecc/.

### encryption

Scrambling information in a way that disguises its meaning. See decryption.

### encryption key

A private key used for encryption only. An encryption key and its equivalent public key, plus a signing key and its equivalent public key, constitute a dual key pair.

### end entity

In a public-key infrastructure (PKI), a person, router, server, or other entity that uses a certificate to identify itself.

### enrollment

The process of requesting and receiving an X.509 certificate for use in a public-key infrastructure (PKI). Also known as *registration*.

### extensions field

See certificate extensions.

## F

### Federal Bridge Certificate Authority (FBCA)

A configuration where two CAs form a circle of trust by issuing cross-pair certificates to each other and storing the two cross-pair certificates as a single certificate pair.

### fingerprint

See certificate fingerprint.

### FIPS PUBS 140-1

Federal Information Standards Publications (FIPS PUBS) 140-1 is a US government standard for implementations of cryptographic modules, hardware or software that encrypts and decrypts data or performs other cryptographic operations, such as creating or verifying digital signatures. Many products sold to the US government must comply with one or more of the FIPS standards. See http://www.itl.nist.gov/div897/pubs/fip140-1.htm.

### firewall

A system or combination of systems that enforces a boundary between two or more networks.

## I

### impersonation

The act of posing as the intended recipient of information sent over a network. Impersonation can take two forms: spoofing and misrepresentation.

### input

In the context of the certificate profile feature, it defines the enrollment form for a particular

certificate profile. Each input is set, which then dynamically creates the enrollment form from all inputs configured for this enrollment.

### intermediate CA

A CA whose certificate is located between the root CA and the issued certificate in a certificate chain.

### IP spoofing

The forgery of client IP addresses.

## J

### JAR file

A digital envelope for a compressed collection of files organized according to the Java™ archive (JAR) format.

### Java™ archive (JAR) format

A set of conventions for associating digital signatures, installer scripts, and other information with files in a directory.

### Java™ Cryptography Architecture (JCA)

The API specification and reference developed by Sun Microsystems for cryptographic services. See http://java.sun.com/products/jdk/1.2/docs/guide/security/CryptoSpec.Introduction.

### Java™ Development Kit (JDK)

Software development kit provided by Sun Microsystems for developing applications and applets using the Java™ programming language.

### Java™ Native Interface (JNI)

A standard programming interface that provides binary compatibility across different implementations of the Java™ Virtual Machine (JVM) on a given platform, allowing existing code written in a language such as C or C++ for a single platform to bind to Java™. See http://java.sun.com/products/jdk/1.2/docs/guide/jni/index.html.

### Java™ Security Services (JSS)

A Java™ interface for controlling security operations performed by Netscape Security Services (NSS).

## K

### KEA

See Key Exchange Algorithm (KEA).

### key

A large number used by a cryptographic algorithm to encrypt or decrypt data. A person's public key, for example, allows other people to encrypt messages intended for that person. The messages must then be decrypted by using the corresponding private key.

### key exchange

A procedure followed by a client and server to determine the symmetric keys they will both use during an SSL session.

### Key Exchange Algorithm (KEA)

An algorithm used for key exchange by the US Government.

## L

### Lightweight Directory Access Protocol (LDAP)

A directory service protocol designed to run over TCP/IP and across multiple platforms. LDAP is a simplified version of Directory Access Protocol (DAP), used to access X.500 directories. LDAP is under IETF change control and has evolved to meet Internet requirements.

### linked CA

An internally deployed certificate authority (CA) whose certificate is signed by a public, third-party CA. The internal CA acts as the root CA for certificates it issues, and the third- party CA acts as the root CA for certificates issued by other CAs that are linked to the same third-party root CA. Also known as "chained CA" and by other terms used by different public CAs.

## M

### manual authentication

A way of configuring a Certificate System subsystem that requires human approval of each certificate request. With this form of authentication, a servlet forwards a certificate request to a request queue after successful authentication module processing. An agent with appropriate privileges must then approve each request individually before profile processing and certificate issuance can proceed.

### MD5

A message digest algorithm that was developed by Ronald Rivest. See also one-way hash.

### message digest

See one-way hash.

### misrepresentation

The presentation of an entity as a person or organization that it is not. For example, a website might pretend to be a furniture store when it is really a site that takes credit-card payments but

never sends any goods. Misrepresentation is one form of impersonation. See also spoofing.

## N

### Netscape Security Services (NSS)

A set of libraries designed to support cross-platform development of security-enabled communications applications. Applications built using the NSS libraries support the Secure Sockets Layer (SSL) protocol for authentication, tamper detection, and encryption, and the PKCS #11 protocol for cryptographic token interfaces. NSS is also available separately as a software development kit.

### nonrepudiation

The inability by the sender of a message to deny having sent the message. A digital signature provides one form of nonrepudiation.

## O

### object signing

A technology that allows software developers to sign Java™ code, JavaScript scripts, or any kind of file and allows users to identify the signers and control access by signed code to local system resources.

### object-signing certificate

A certificate that's associated private key is used to sign objects using the technology known as object signing.

### OCSP

Online Certificate Status Protocol.

### one-way hash

1. A number of fixed-length generated from data of arbitrary length with the aid of a hashing algorithm. The number, also called a message digest, is unique to the hashed data. Any change in the data, even deleting or altering a single character, results in a different value.

2. The content of the hashed data cannot be deduced from the hash.

### operation

The specific operation, such as read or write, that is being allowed or denied in an access control instruction.

### output

In the context of the certificate profile feature, it defines the resulting form from a successful certificate enrollment for a particular certificate profile. Each output is set, which then dynamically creates the form from all outputs configured for this enrollment.

## P

### password-based authentication

Confident identification by means of a name and password. See also authentication, certificate-based authentication.

### PKCS #10

The public-key cryptography standard that governs certificate requests.

### PKCS #11

The public-key cryptography standard that governs cryptographic tokens such as smart cards.

### PKCS #11 module

A driver for a cryptographic device that provides cryptographic services, such as encryption and decryption, through the PKCS #11 interface. A PKCS #11 module, also called a *cryptographic module* or *cryptographic service provider*, can be implemented in either hardware or software. A PKCS #11 module always has one or more slots, which may be implemented as physical hardware slots in some form of physical reader, such as for smart cards, or as conceptual slots in software. Each slot for a PKCS #11 module can in turn contain a token, which is the hardware or software device that actually provides cryptographic services and optionally stores certificates and keys. Red Hat provides a built-in PKCS #11 module with Certificate System.

### PKCS #12

The public-key cryptography standard that governs key portability.

### PKCS #7

The public-key cryptography standard that governs signing and encryption.

### private key

One of a pair of keys used in public-key cryptography. The private key is kept secret and is used to decrypt data encrypted with the corresponding public key.

### proof-of-archival (POA)

Data signed with the private Data Recovery Manager transport key that contains information about an archived end-entity key, including key serial number, name of the Data Recovery Manager, subject name of the corresponding certificate, and date of archival. The signed proof-of-archival data are the response returned by the Data Recovery Manager to the Certificate Manager after a successful key archival operation. See also Data Recovery Manager transport certificate.

### public key

One of a pair of keys used in public-key cryptography. The public key is distributed freely and

published as part of a certificate. It is typically used to encrypt data sent to the public key's owner, who then decrypts the data with the corresponding private key.

### public-key cryptography

A set of well-established techniques and standards that allow an entity to verify its identity electronically or to sign and encrypt electronic data. Two keys are involved, a public key and a private key. A public key is published as part of a certificate, which associates that key with a particular identity. The corresponding private key is kept secret. Data encrypted with the public key can be decrypted only with the private key.

### public-key infrastructure (PKI)

The standards and services that facilitate the use of public-key cryptography and X.509 v3 certificates in a networked environment.

## R

### RC2, RC4

Cryptographic algorithms developed for RSA Data Security by Rivest. See also cryptographic algorithm.

### Red Hat Certificate System

A highly configurable set of software components and tools for creating, deploying, and managing certificates. Certificate System is comprised of five major subsystems that can be installed in different Certificate System instances in different physical locations: Certificate Manager, Online Certificate Status Manager, Data Recovery Manager, Token Key Service, and Token Processing System.

### registration

See enrollment.

### root CA

The certificate authority (CA) with a self-signed certificate at the top of a certificate chain. See also CA certificate, subordinate CA.

### RSA algorithm

Short for Rivest-Shamir-Adleman, a public-key algorithm for both encryption and authentication. It was developed by Ronald Rivest, Adi Shamir, and Leonard Adleman and introduced in 1978.

### RSA key exchange

A key-exchange algorithm for SSL based on the RSA algorithm.

## S

### sandbox

A Java™ term for the carefully defined limits within which Java™ code must operate.

### Secure Sockets Layer (SSL)

A protocol that allows mutual authentication between a client and server and the establishment of an authenticated and encrypted connection. SSL runs above TCP/IP and below HTTP, LDAP, IMAP, NNTP, and other high-level network protocols.

### self tests

A feature that tests a Certificate System instance both when the instance starts up and on-demand.

### server authentication

The process of identifying a server to a client. See also client authentication.

### server SSL certificate

A certificate used to identify a server to a client using the Secure Sockets Layer (SSL) protocol.

### servlet

Java™ code that handles a particular kind of interaction with end entities on behalf of a Certificate System subsystem. For example, certificate enrollment, revocation, and key recovery requests are each handled by separate servlets.

### SHA-1

Secure Hash Algorithm, a hash function used by the US government.

### signature algorithm

A cryptographic algorithm used to create digital signatures. Certificate System supports the MD5 and SHA-1 signing algorithms. See also cryptographic algorithm, digital signature.

### signed audit log

See audit log.

### signing certificate

A certificate that's public key corresponds to a private key used to create digital signatures. For example, a Certificate Manager must have a signing certificate that's public key corresponds to the private key it uses to sign the certificates it issues.

### signing key

A private key used for signing only. A signing key and its equivalent public key, plus an encryption key and its equivalent public key, constitute a dual key pair.

**single sign-on**

1. In Certificate System, a password that simplifies the way to sign on to Red Hat Certificate System by storing the passwords for the internal database and tokens. Each time a user logs on, he is required to enter this single password.

2. The ability for a user to log in once to a single computer and be authenticated automatically by a variety of servers within a network. Partial single sign-on solutions can take many forms, including mechanisms for automatically tracking passwords used with different servers. Certificates support single sign-on within a public-key infrastructure (PKI). A user can log in once to a local client's private-key database and, as long as the client software is running, rely on certificate-based authentication to access each server within an organization that the user is allowed to access.

**slot**

The portion of a PKCS #11 module, implemented in either hardware or software, that contains a token.

**smart card**

A small device that contains a microprocessor and stores cryptographic information, such as keys and certificates, and performs cryptographic operations. Smart cards implement some or all of the PKCS #11 interface.

**spoofing**

Pretending to be someone else. For example, a person can pretend to have the email address **`jdoe@example.com`**, or a computer can identify itself as a site called **`www.redhat.com`** when it is not. Spoofing is one form of impersonation. See also misrepresentation.

**SSL**

See Secure Sockets Layer (SSL).

**subject**

The entity identified by a certificate. In particular, the subject field of a certificate contains a subject name that uniquely describes the certified entity.

**subject name**

A distinguished name (DN) that uniquely describes the subject of a certificate.

**subordinate CA**

A certificate authority that's certificate is signed by another subordinate CA or by the root CA. See CA certificate, root CA.

**symmetric encryption**

An encryption method that uses the same cryptographic key to encrypt and decrypt a given

message.

## T

**tamper detection**

A mechanism ensuring that data received in electronic form entirely corresponds with the original version of the same data.

**token**

A hardware or software device that is associated with a slot in a PKCS #11 module. It provides cryptographic services and optionally stores certificates and keys.

**tree hierarchy**

The hierarchical structure of an LDAP directory.

**trust**

Confident reliance on a person or other entity. In a public-key infrastructure (PKI), trust refers to the relationship between the user of a certificate and the certificate authority (CA) that issued the certificate. If a CA is trusted, then valid certificates issued by that CA can be trusted.

## V

**virtual private network (VPN)**

A way of connecting geographically distant divisions of an enterprise. The VPN allows the divisions to communicate over an encrypted channel, allowing authenticated, confidential transactions that would normally be restricted to a private network.

# Index

## A